E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Diseño e implementación de plugins para la transformación de señales Ambisonic

Grado en Ingeniería
en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Autor: Rafael Marín Hernández
Tutor: Ricardo San Martín Murugarren
Fecha: junio 2020

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

ÍNDICE

1. INTRODUCCIÓN Y OBJETO

El sonido envolvente, que trabaja en todo el área alrededor del oyente, es un tema de investigación que ha cobrado una gran importancia durante los últimos años, especialmente con el nacimiento de las tecnologías de los sistemas de realidad virtual y grabación y reproducción de video en 360 grados, respaldadas por Google, Facebook y Youtube, por lo que el estudio de estos campos se encuentra en auge.

La mayoría de los estándares actuales para la presentación de audio espacial se basan en configuraciones de altavoces específicas, como serían por ejemplo las instalaciones 5.1 o 7.1. En estos sistemas, las señales están formadas por varios canales que deben ser enviados por separado a cada uno de los altavoces concretos (Channel Based Audio). En contraste a esto, existen tecnologías (Object Based Audio, Scene Based Audio) capaces de grabar, sintetizar y reproducir campos sonoros de forma independiente al sistema de presentación que se vaya a utilizar, lo que ofrece una gran flexibilidad para la reproducción de estas escenas en auriculares o cualquier configuración de altavoces.

Uno de los principales y más versátiles estándares para caracterizar un campo sonoro completo es Ambisonics. Esta tecnología fue desarrollada en origen por Michael Gerzon en la década de 1970 en la Universidad de Oxford, donde introdujo las bases para el primer orden de este estándar. Posteriormente se extendió esta teoría a Ambisonics de órdenes superiores, consiguiendo así una representación mas detallada de la imagen sonora.

Al trabajar con este tipo de codificación se hace evidente la necesidad de realizar transformaciones a estas señales para adaptarlas a los diferentes sistemas de reproducción, mezclarlas entre ellas o con otro tipo de señales, y adecuarlas para que tengan una calidad de audio suficientemente buena como para ser presentadas ante un espectador. Para este propósito existen varias suites de plugins de efectos de audio desarrolladas por diferentes grupos de investigación que permiten realizar los procesos básicos que deben sufrir las señales de audio para su mezcla o adaptación a unos estándares de calidad, además de las transformaciones que necesitan aplicarse para que puedan ser reproducidas en diferentes configuraciones de altavoces o adaptadas a auriculares.

Por otro lado muchos de estos efectos de audio están orientados también a la síntesis o creación de escenas sonoras a través de la codificación de señales mono o estéreo, colocándolas en el punto deseado del campo que rodea al oyente, consiguiendo así crear nuevas producciones envolventes de audio a partir de señales simples.

1.1. Sonorizarte

Fig. 1.1. – Logotipo del proyecto Sonorizarte

Este proyecto se enmarca dentro de un proyecto mayor, llamado Sonorizarte, desarrollado por el grupo de investigación Acústica de la UPNA, cuyo objetivo final es la creación de una

instalación sonora donde puedan experimentarse escenas de audio tridimensionales. Esta instalación será capaz de trabajar en forma de exposición, mostrando creaciones realizadas por distintos compositores o artistas sonoros y, además, funcionar en modo interactivo, donde el público podrá crear en tiempo real escenas o paisajes sonoros a través de interfaces de interacción con el usuario desarrolladas específicamente para este proyecto.

La instalación se utilizará también para la exposición de escenas sonoras creadas a partir de sonidos de la naturaleza y de entornos y actividades pertenecientes al patrimonio cultural y arquitectónico, lo que permitirá la recreación de estos entornos y paisajes de forma inmersiva y el análisis auditivo de los mismos. Para este propósito se contará con la colaboración de diferentes investigadores en paisajes sonoros.

1.2. Objeto

Normalmente los plugins que trabajan con señales Ambisonic implementados hasta la fecha se basan en realizar la codificación y decodificación de estas señales, en modificar la escena sonora, aplicándole rotación, espejado o warping, o en transformaciones asociadas a procesos clásicos de producción de audio, como serian la ecualización, compresión, reverb…

El objetivo de este proyecto es el diseño, desarrollo e implementación de nuevo software, en forma de plugins de efectos de audio que permitan realizar transformaciones de estos campos sonoros de formas creativas y experimentales no implementadas hasta ahora. El software desarrollado será de utilidad para la creación y producción de señales Ambisonic que posteriormente podrán ser presentadas en la instalación sonora del proyecto Sonorizarte.

Para poder llevar a cabo este proyecto es importante el estudio y comprensión de los textos y artículos publicados hasta ahora dentro del campo de investigación de las señales Ambisonic, además del conocimiento y especialización en los entornos de programación orientados a plugins de audio con los que se está trabajando actualmente.

2. AMBISONICS

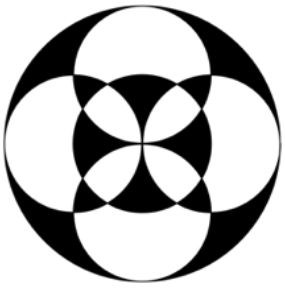

Fig. 2.1. – Icono del estándar Ambisonic

El estándar Ambisonics se basa en los principios de grabación de audio con micrófonos coincidentes establecidos en la década de 1930. Este formato de grabación se fundamenta en configuraciones de micrófonos direccionales dentro de un espacio lo más pequeño posible. De esta manera, el sonido incidente llega aproximadamente con el mismo retraso a todos los micrófonos del sistema. En estos principios se basan los procesos de grabación XY y MS. Así se pueden realizar grabaciones estéreo basadas en la intensidad que pueden ser reproducidas en una pareja de altavoces estéreo.

En este ámbito aparecen los primeros efectos para la transformación del campo sonoro a través del procesado Mid-Side, que consiguen aumentar o disminuir el ancho del campo estéreo.

En la década de 1970 se establecen los principios para grabación y reproducción de señales Ambisonics de primer orden, consiguiendo así realizar una codificación de las características del campo sonoro en dos dimensiones. En muchas de las aplicaciones actuales de realidad virtual y reproducción 360 se sigue utilizando este primer orden ya que, además de poder ser reproducido en diferentes configuraciones de altavoces, permite ser presentado de forma interactiva en auriculares con sistemas de seguimiento de la cabeza para que la escena sonora sea estática para el oyente.

La codificación de campos sonoros con Ambisonics de primer orden en dos dimensiones tiene la ventaja de que se puede hacer con pocos micrófonos y conseguir una alta calidad, pero la reproducción en altavoces no suele ser convincente, ya que no existe suficiente resolución, además de que solo es aplicable a configuraciones bidimensionales, por lo que es necesario trabajar en órdenes superiores para obtener mejoras en la representación de la imagen sonora envolvente.

2.1. Sistema de coordenadas

El sistema de coordenadas que se utiliza al trabajar en el ámbito de Ambisonics se define de la forma mostrada en la figura 2.2. (Kronlachner 2019).

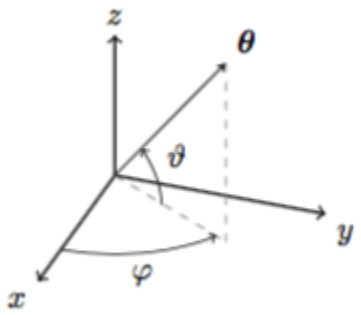

Fig. 2.2 – Sistema de coordenadas

Desde la perspectiva del oyente, el eje x apunta al frente, el eje y a la izquierda y el eje z hacia la parte superior. Sin embargo, dentro de Ambisonics se suelen utilizar para la representación las coordenadas esféricas, donde φ es el ángulo de azimut y ϑ el de elevación. En muchos casos será necesario el uso de ambos sistemas de coordenadas, esféricas y cartesianas, por lo que será útil tener presente la forma en que se realiza la conversión de unas a otras indicada en las expresiones (1) y (2).

$$\theta = \begin{pmatrix} \theta_x \\ \theta_y \\ \theta_z \end{pmatrix} = \begin{pmatrix} \cos(\varphi)\cos(\vartheta) \\ \text{sen}(\varphi)\cos(\vartheta) \\ \text{sen}(\vartheta) \end{pmatrix} \qquad (1)$$

$$\varphi = -\arctan\left(\frac{\theta_y}{\theta_x}\right) \qquad \vartheta = \arctan\left(\frac{\theta_z}{\sqrt{\theta_x^2 + \theta_y^2}}\right) \qquad (2)$$

2.2. Armónicos esféricos

Para la descripción de las señales Ambisonic se utilizan funciones armónicas continuas de resolución escalable. En el caso tridimensional los distintos armónicos esféricos forman un conjunto de funciones básicas ortogonales que pueden usarse para describir cualquier función en la superficie de una esfera. Una señal de audio $f(t)$ que llega desde una dirección determinada $\theta = (\varphi, \vartheta)$ se puede considerar como una señal de audio envolvente $f(\varphi, \vartheta, t)$, esta se puede expresar mediante su expansión en armónicos esféricos hasta un orden de truncamiento N mediante la ecuación (3) (Zotter 2019).

$$f(\varphi, \vartheta, t) = \sum_{n=0}^{N} \sum_{m=-n}^{n} Y_n^m(\varphi, \vartheta) \phi_{nm}(t) \qquad (3)$$

Donde Y_n^m corresponde a los armónicos esféricos de orden n y grado m mientras que ϕ_{nm} hace referencia los coeficientes de expansión. Conforme se aumenta el orden N se consigue una representación espacial mucho mas precisa. En la siguiente figura, se pueden observar los diferentes armónicos esféricos hasta orden N = 4, cada uno correspondiente a un orden n y grado m concretos, siguiendo la numeración de canales ACN (Ambisonic Channel Numbering).

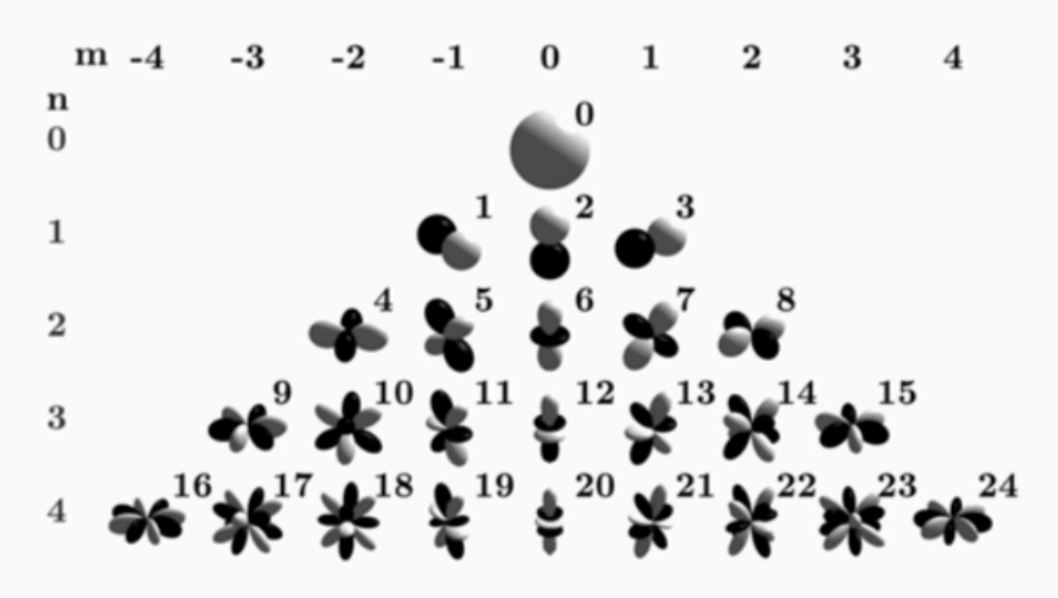

Fig. 2.3 – Representación gráfica de armónicos esféricos hasta el cuarto orden

La numeración de canales ACN, establece una secuencia para los armónicos esféricos como se define en (4) (Kronlachner 2019).

$$ACN = n^2 + n + m \qquad (4)$$

Esta secuencia se puede descifrar fácilmente y ser utilizada para obtener el orden del armónico esférico como se muestra en (5).

$$n = \sqrt{ACN} \qquad (5)$$

Y también es muy sencillo obtener su grado con de la forma indicada la expresión (6).

$$m = ACN - n^2 - n \qquad (6)$$

2.3. Codificación y decodificación en 3D

Para codificar una señal s en señales Ambisonic χ_{nm}, se debe multiplicar la señal por el codificador, que representa la dirección θ_s de la señal para los pesos $Y_n^m(\theta_s)$, como se muestra en la expresión (7) (Zotter 2019).

$$\chi_{nm}(t) = Y_n^m(\theta_s)s(t) \qquad (7)$$

En notación vectorial (8).

$$\chi_N = \mathrm{y}_N(\theta_s)s \qquad (8)$$

Donde $\mathrm{y}_N = [Y_0^0(\theta_s), Y_1^{-1}(\theta_s), \dots Y_N^N(\theta_s)]^T$ es un vector columna de $(N+1)^2$ componentes. Las señales Ambisonic en χ_N están ponderadas por un vector de pesos para la supresión de los lóbulos laterales $\mathrm{a}_N = [\mathrm{a}_0, \mathrm{a}_1, \mathrm{a}_1, \mathrm{a}_1, \mathrm{a}_2 \dots, \mathrm{a}_N]^T$ expresado en forma de

matriz diagonal $diag\{a_N\}$ y luego decodificado a las señales x correspondientes a los L altavoces por un decodificador de muestreo. Esto se puede ver en la ecuación (9).

$$D = \sqrt{\frac{4\pi}{L}}[y_N(\theta_1), \dots, y_N(\theta_L)]^T = \sqrt{\frac{4\pi}{L}} Y_N^T \qquad (9)$$

Obteniendo con su aplicación las señales x en la formula (10).

$$x = Ddiag\{a_N\}\chi_N \qquad (10)$$

El sistema para codificar y decodificar, puede ser descrito como un conjunto de ganancias para los altavoces de una fuente virtual expresado en (11).

$$g = Ddiag\{a_N\}y_N(\theta_s) \qquad (11)$$

En la aplicación especifica de un decodificador de muestreo para 3D.

$$g = \sqrt{\frac{4\pi}{L}} Y_N^T diag\{a_N\}y_N(\theta_s) \qquad (12)$$

2.4. T-designs

Antes de analizar la decodificación de Ambisonics para altavoces y los procesos de transformación de las señales Ambisonic es necesario introducir el concepto de t-designs esféricos que son diferentes discretizaciones posibles preestablecidas para una esfera. En el caso de Ambisonics es necesario que si se trabaja en un orden N se utilice un t-diseño con t mayor o igual a 2N, donde t es el numero de muestras que se toman de la esfera.

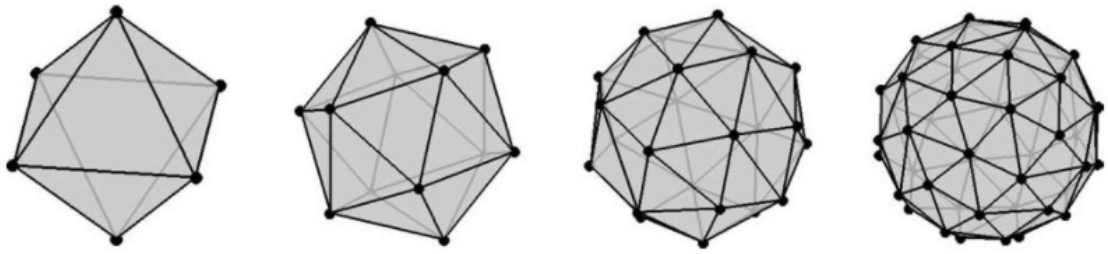

Fig. 2.4. – Distintos t-designs aplicables para realizar el muestreo de una esfera

En el caso del proyecto sonorizarte, se utilizará un t-design de 24 puntos, ya que la configuración de la esfera contará con 24 altavoces situados en estos puntos.

2.5. Decodificación de señales Ambisonic para reproducción en altavoces

La decodificación de Ambisonics para altavoces cuenta con varios problemas, en su primer orden no proporciona resultados muy estables, en ordenes superiores el éxito depende en gran medida de lo uniforme que sea la distribución de altavoces y esta comprobado que el uso de demasiados altavoces tiene un efecto degradante en la señal.

El uso de t-designs permite utilizar una decodificación SAD (Sampling Ambisonic Decoder) que trabaje sin la aparición de efectos indeseados como son fluctuaciones o errores de mapeo

direccional, ya que se realiza una discretización regular de la esfera. Por lo que el diseño de la esfera utilizada en el proyecto sonorizarte será una distribución de altavoces en forma de t-design.

2.6. Decodificación de señales Ambisonic para reproducción en auriculares

La decodificación de Ambisonics para auriculares se puede hacer de forma similar a la de altavoces, la variación principal en el sistema de decodificación, es que las señales correspondientes a cada altavoz se transmiten a los auriculares por medio de convolución con las respuestas al impulso en relación a la cabeza, HRIR (Head Realted Impulse Response), para las direcciónes de reproducción correspondientes.

Este método de decodificación se basa en el uso de una configuración de altavoces virtuales o imaginarios situados virtualmente en unas direcciones concretas, para las que se realiza una decodificación de la señal, tras esto las señales obtenidas para cada uno de estos altavoces es convolucionada con la HRIR correspondiente a esa dirección. Existen numerosas bases de datos donde obtener HRIR de diferentes sujetos y micrófonos binaurales para utilizarlas en este proceso.

2.7. Flujo de señal

En este apartado se describe la ruta que debe seguir la señal de audio para realizar una producción Ambisonic, desde el proceso de codificación hasta su decodificación. Esta estructura de procesado es independiente de la estación de trabajo para el procesamiento de audio utilizada, ya que en todas se puede realizar la edición de audio de forma similar. La limitación vendrá dada por el tamaño de bus que pueda utilizar, ya que cada orden Ambisonic exige un número mínimo de canales.

En el siguiente diagrama de bloques se muestra el camino que debe seguir la señal de audio envolvente para ser procesada (Zotter 2019).

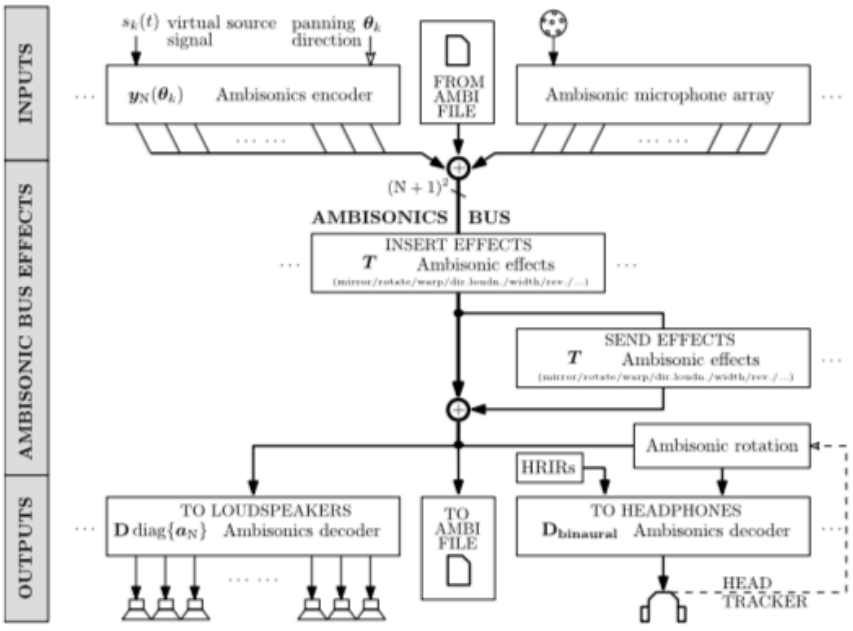

Fig. 2.5. – Esquema básico de ruteo para trabajar con señales y efectos de audio Ambisonic

El procesado se divide en tres secciones principales:

En la parte de entradas se introducen en el bus Ambisonic las señales a procesar. Para ello, si son señales monofónicas, estereofónicas o multicanal (no Ambisonic), deberán ser codificadas con uno de los plugins disponibles, tales como StereoEncoder y MultiEncoder de la suite IEM o sparta_ambiENC de la suite Sparta. Estos permitirán convertir estas señales al ámbito Ambisonics, además de posicionarlas en el punto que interese de la esfera envolvente. Por otro lado, podrían introducirse señales Ambisonic directamente desde archivos previamente almacenados o procedentes de grabaciones de configuraciones de micrófonos específicas para grabar audio espacial, siempre y cuando cumplan los requisitos de codificación de Format B. Estas señales deberán coincidir en orden con el que vayamos a utilizar en el bus de procesado, ya que esto determina el numero de canales que utilizan. En el caso del proyecto Sonorizarte se utilizara el tercer orden Ambisonic, cuyas señales están formadas por 16 canales. Si se quisiese utilizar una señal Ambisonic de orden menor o mayor, se debería realizar una conversión de orden con ayuda del plugin COMPASS_Upmixer de la casa Sparta.

En la parte del bus de efectos Ambisonic la señal ya cumple con los requisitos del orden en el que se está trabajando, por lo que el número de canales es el correcto. Para realizar un procesado existen dos opciones, al igual que en el ámbito de la edición de audio tradicional se pueden utilizar los efectos como inserto en el propio bus o como envío-retorno. En el caso de utilizar el efecto como inserto afectará por completo a toda la señal, dando como resultado únicamente la señal transformada. Para ello solo habría que cargar el plugin del efecto deseado en el bus Ambisonic y ajustar los parámetros que permita transformar. Por otro lado, utilizando el otro enfoque se podría realizar un envío de toda la señal a otro bus con el número de canales necesario, con esto conseguiremos tener la misma señal Ambisonic discurriendo por dos buses de audio, en uno de ellos la señal se dejara intacta, en el otro, donde estará insertado el efecto, la señal sufrirá la transformación deseada, una vez realizado este proceso la señal procesada regresara al bus original donde se encuentra la señal sin procesar, con esto se combinaran ambas señales. Para obtener la sensación de una aplicación mayor o menor del efecto según los intereses del procesado, se podrá trabajar con la ganancia de la pista procesada para que el efecto sea mas o menos evidente.

Por último, en la sección de salida se puede realizar la decodificación de la señal para adaptarla a su reproducción en una configuración de altavoces específica. Para ello se pasaría toda la señal del bus Ambisonic por uno de los plugins de decodificación, como podría ser SimpleDecoder de IEM, donde podemos cargar una configuración de altavoces previamente diseñada con AllRADecoder, o sparta_ambiDEC, donde podemos establecer directamente la posición de cada uno de los altavoces. También nos podría interesar exportar directamente el audio de bus como un archivo de señal Ambisonic directamente, lo que sería posible con las opciones de exportación del DAW que se esté utilizando. Como último caso se podría desear la reproducción de la señal en auriculares. En este caso habría que realizar la decodificación correspondiente a un sistema binaural, convolucionando con las HRTFs correspondientes, con uno de los plugins disponibles. Estos son BinauralDecoder de la suite IEM o sparta_ambiBIN de Sparta. Además, se podría utilizar un sistema de tracking para la cabeza que enviaría mensajes OSC a estos plugins para hacer un seguimiento del movimiento del usuario y adaptar la señal en consecuencia.

2.8. Creación de efectos mediante transformaciones de señales Ambisonics

Los efectos básicos para la aplicación en señales Ambisonic son independientes de la frecuencia y se basan en dos principios: el mapeo direccional, que incluye espejado, rotación y deformación de la escena; y la re-ponderación direccional, que consiste en la modificación del nivel en función de la dirección dentro de la escena.

Sea $f(\theta, t)$ una señal surround, entendida ésta como un conjunto de datos distribuidos sobre la superficie de una esfera que varían en el tiempo. Matemáticamente podemos describir las transformaciones que sufre esta señal con la ecuación (13) (Zotter 2019).

$$f'(\theta', t) = g(\theta) f(\theta, t) \qquad (13)$$

La ecuación (13) expresa una operación que puede seleccionar cada dirección de entrada, ponderar su señal con una ganancia direccional, $g(\theta)$, y volver a asignarla a una nueva dirección, $\theta' = \tau\{\theta\}$. De esa manera, la señal original $f(\theta, t)$ se convierte en la señal transformada $f'(\theta', t)$'.

Esta expresión puede escribirse en el dominio de señales Ambisonic de la forma indicada en (14).

$$\chi'_{N'}(t) = T\chi_N(t) \qquad (14)$$

La señal transformada será la entrada Ambisonic modificada por la matriz T de transformación, que contiene coeficientes que se aplicarán a los distintos armónicos esféricos que conforman la señal, pudiendo incluso la transformación requerir un orden Ambisonic superior para ejecutarse correctamente Por tanto, cada efecto deseado consistirá en aplicar la matriz de transformación correspondiente.

2.8.1. Espejado

Para conseguir el efecto de espejado, donde se refleja completamente la señal envolvente sobre uno de los ejes, no es necesario realizar el re-mapeo y únicamente es necesario invertir la fase de las señales correspondientes a los armónicos esféricos con simetría impar sobre ese eje.

En base a esto, la matriz de transformación será diagonal con la secuencia correspondiente a los cambios de signo necesarios, $T = diag\{\mathrm{c}\}$ siendo c la secuencia de coeficientes a aplicar. Podemos conseguir tres tipos de transformaciones de espejado:

Espejado Arriba-Abajo. Para obtener este efecto se deben invertir los signos de armónicos esféricos simétricos impares con respecto al eje z. Se pueden hallar fácilmente los coeficientes que irán en la diagonal de la matriz de transformación con la ecuación (15).

$$c_{n,m} = (-1)^{n+m} \qquad (15)$$

De esta forma los armónicos esféricos que sufrirán la inversión serán los marcados en la siguiente figura 2.6., donde el número situado al lado del armónico corresponde al número de canal al que habrá que invertir la fase:

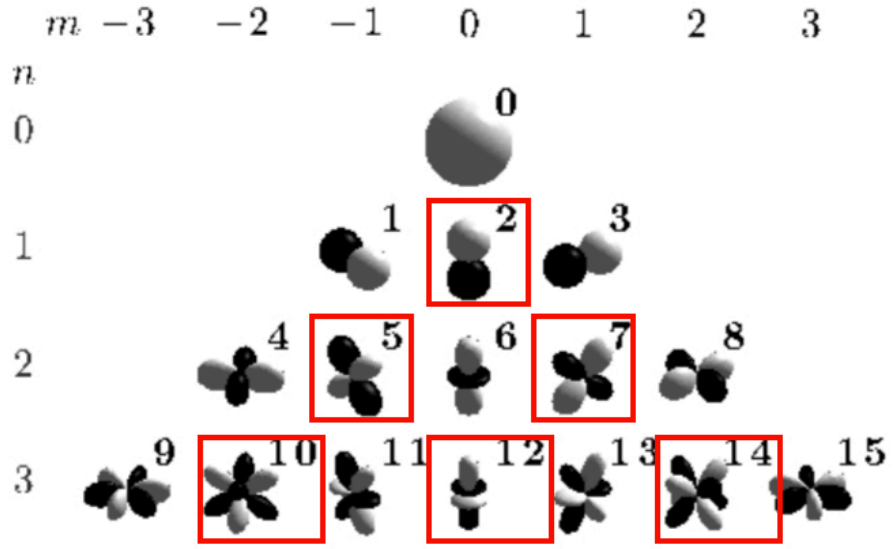

Fig. 2.6. – Armónicos esféricos a los que se debe aplicar un cambio de signo para el espejado Up-Down

Espejado Izquierda-Derecha. Mediante un razonamiento similar, se invertirán los signos de los armónicos esféricos correspondientes. Los coeficientes vienen dados por la ecuación (16).

$$c_{n,m} = (-1)^{m<0} \qquad (16)$$

Los armónicos esféricos que se verán afectados por el cambio de signo serán los siguientes, mostrados en 2.7..

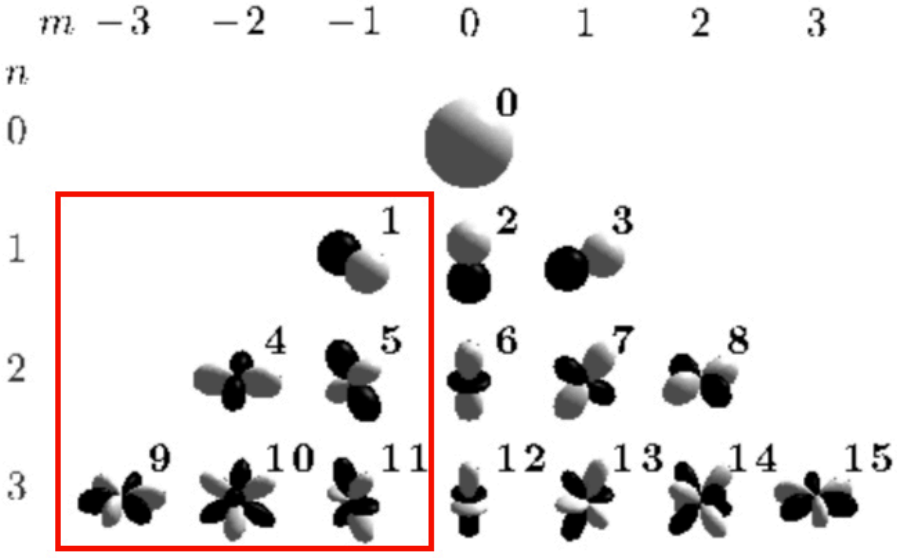

Fig. 2.7. – Armónicos esféricos a los que se debe aplicar un cambio de signo para el espejado Left-Right

Espejado Frontal-Trasero: También se invierten los signos de los armónicos esféricos correspondientes, dados los coeficientes por la expresión (17).

$$c_{n,m} = (-1)^{m+(m<0)} \qquad (17)$$

En este caso los armónicos que sufren la inversión de signo son los siguientes se muestran en la figura 2.8..

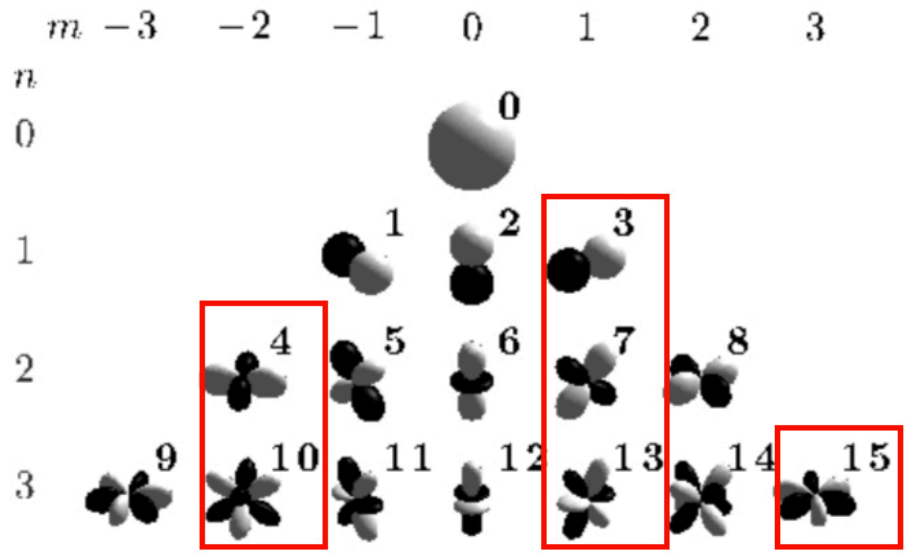

Fig. 2.8. – Armónicos esféricos a los que se debe aplicar un cambio de signo para el espejado Front-Back

2.8.2. Rotación sobre el eje Z

La rotación sobre el eje vertical, es un caso particular de la rotación tridimensional que se puede simplificar mucho con una implementación más sencilla y que requiere un menor coste computacional a la hora de realizarla en forma de software, consiguiendo hacer una rotación virtual alrededor del oyente que auditivamente es muy evidente. Se puede realizar fácilmente configurando las matrices de transformación de la siguiente forma mostrada en (28).

$$R(m,\varphi) = \begin{bmatrix} \cos(m\varphi) & sen(m\varphi) \\ -sen(m\varphi) & \cos(m\varphi) \end{bmatrix} \qquad (18)$$

Siendo φ el ángulo de rotación deseado. La matriz será aplicada a pares de señales entre los que cambia el signo del grado. Para ello se deberá proceder de la siguiente manera expresada en la fórmula (19).

$$\begin{bmatrix} \phi_{-m}(\varphi_s + \varphi) \\ \phi_m(\varphi_s + \varphi) \end{bmatrix} = R(m,\varphi) * \begin{bmatrix} \phi_{-m}(\varphi_s) \\ \phi_{-m}(\varphi_s) \end{bmatrix} \qquad (19)$$

Siendo en ella φ_s la angulación original respecto al eje z y ϕ las señales correspondientes a los armónicos esféricos.

2.8.3. Rotación en 3 dimensiones

Como las rotaciones en el eje z son mucho más sencillas de implementar, para rotaciones que involucran a los demás ejes, se suele hacer una combinación de rotaciones variables en el eje z junto con rotaciones fijas de 90 grados en el eje y. La matriz de rotación genérica, a partir de la que se pueden obtener las rotaciones con ángulo variable en el eje y o con ángulo fijo de 90 grados en este eje, se obtiene a partir de la formula (20).

$$R(\phi,\theta,\psi)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos(\phi) & -sen(\phi)\\ 0 & sen(\phi) & \cos(\phi)\end{bmatrix}\begin{bmatrix}\cos(\theta) & 0 & sen(\theta)\\ 0 & 1 & 0\\ -sen(\theta) & 0 & \cos(\theta)\end{bmatrix}\begin{bmatrix}\cos(\psi) & -sen(\psi) & 0\\ sen(\psi) & \cos(\psi) & 0\\ 0 & 0 & 1\end{bmatrix} \quad (20)$$

Esta matriz de R se debe multiplicar por el vector de coordenadas cartesianas de la posición original para realizar el re-mapeo direccional mencionado al principio del capitulo de la forma expresada en (21).

$$\theta' = R(\phi,\theta,\psi)\begin{pmatrix}\theta_x\\ \theta_y\\ \theta_z\end{pmatrix} \quad (21)$$

Este re-mapeo direccional se aplica a la señal Ambisonic a través de la matriz T que se formará según la expresión (22).

$$T = diag\{b\}Y^T(\Theta_t)diag\{g(\Theta_t)\}Y(\mathfrak{T}^{-1}\{\Theta_t\}) \quad (22)$$

Donde Y es un vector formado por los armónicos esféricos discretizados en base a un t-design, g es el vector de ganancias para cada una de las direcciones, también discretizado por un t-design, que en este caso será 1 para todas las direcciones y b vendrá dado por la ecuación (23.)

$$b = [\frac{2n_{ACN}+1}{L}]_{ACN} \quad (23)$$

Donde L es el numero total de canales correspondientes al orden en el que se esta trabajando.

3. PROGRAMACIÓN DE EFECTOS DE AUDIO

3.1. Software de efectos de audio

La forma mas común de trabajar en el ámbito de la producción, edición y procesado de audio actualmente es utilizando una estación de trabajo de audio digital o DAW (Digital Audio Workstation), un sistema en forma de software que permite realizar grabación y edición de audio digital. Normalmente una plataforma DAW contiene todos los elementos y efectos necesarios para realizar correctamente una producción de audio, pero lo más habitual es utilizar componentes de terceros para generar y procesar el sonido. En este último caso el DAW funciona como host, alojando y dando soporte a otro software en el que delega ciertas tareas. En este punto entran en juego los plugins de audio.

Normalmente un plugin es un software que agrega nuevas funcionalidades a la plataforma o programa que lo está alojando, en este caso al DAW. Dentro del entorno del procesado de audio, los plugins se dividen en tres familias principales que serian:

Instrumentos virtuales. donde se incluye cualquier plugin que genera audio por sí mismo.

Efectos. que engloban a todo tipo de procesadores dinámicos, como podrían ser compresores, reverberaciones, delays….

Herramientas. que incluyen todo tipo de plugins de medición y análisis.

Existen varios formatos específicos para desarrollar plugins de audio, siendo los más habituales VST, AU y VST3.

3.1.1. VST

VST son las siglas de Virtual Studio Technology, un estándar desarrollado por la compañía Steinberg para conectar plugins de audio al software que actúa como host. Es el formato más utilizado para el desarrollo de plugins, ya que fue el primer formato en liberarse y convertirse en gratuito, por lo que se ha convertido en el estándar de la industria de edición de audio.

3.1.2. AU

Estas siglas corresponden a Audio Unit, el estándar de Apple desarrollado para la arquitectura de plugins. Es un formato equivalente a VST, ya que su estructura es muy similar por lo que la mayoría de veces, al ser fácil la compilación en ambos formatos, los plugins cuentan con su propia versión VST y Audio Unit.

3.1.3. VST3

Introduce una actualización, nuevas características y grandes mejoras al formato VST, convirtiéndose en el nuevo estándar para plugins de audio.

Las principales mejoras que introduce frente al formato VST son: entradas y salidas dinámicas, haciendo que el plugin se adapte a la pista o bus donde está trabajando, consiguiendo una mayor flexibilidad; activación y desactivación de buses en VSTs que cuenten con múltiples salidas; interfaz de usuario re-escalable; automatización de sus parámetros en el DAW de forma más precisa; organización lógica de los parámetros en

estructura de árbol; mayor integración con controladores remotos hardware a través de mensajes MIDI; múltiples entradas y salidas MIDI; y procesamiento a 64 bits.

3.2. Juce

Fig. 3.1. – Logotipo del framework Juce

Juce [1] es una de las plataformas más utilizadas para programación y desarrollo de plugins de audio (Robinson 2013). Se trata de un marco de referencia (framework) de código abierto y multiplataforma, realizado sobre C++, que contiene un conjunto de librerías, funciones y clases ya implementadas que simplifican el diseño y desarrollo de las aplicaciones. Cuenta con presets diseñados para la programación de distintos tipos de software. Estas plantillas permiten dar un mejor soporte para realizar aplicaciones GUI, aplicaciones de audio, aplicaciones animadas, aplicaciones OpenGL, aplicaciones de consola y plugins de audio.

Uno de los objetivos de Juce es que el software se escriba de forma que un mismo código se pueda compilar y ejecutar de forma sencilla en distintas plataformas, como son Windows, Mac OSX y Linux. A la hora de la exportación, Juce permite realizarla de manera sencilla para diferentes formatos, como son VST, VST3 y AU.

Juce fue desarrollado originalmente para desarrollo de aplicaciones de audio, por lo que su librería de funciones orientadas al procesado de sonido es muy amplia, y da soporte a distintos dispositivos de audio como CoreAudio, Asio, sintetizadores o dispositivos MIDI. Esto ha hecho que su uso se haya generalizado en el entorno del desarrollo de plugins de audio.

Las plantillas con las que cuenta Juce son las explicadas a continuación, permitiendo generar fácilmente cualquiera de estos tipos de aplicaciones.

Aplicación GUI. Es el tipo mas genérico de todos los proyectos que utilizan un interfaz gráfico de usuario, sus propósitos pueden ser muy variados. Esta plantilla se usa cuando se está seguro de que se necesita un interfaz gráfico de usuario pero no están completamente definidas la flexibilidad o funcionalidades que debe implementar el software.

Aplicación de audio. Es similar a la aplicación GUI, pero se basa en clases orientadas a realizar el procesado de audio. Se debe usar esta plantilla cuando se quiera realizar una aplicación standalone que utilice un interfaz gráfico de usuario y además requiera funcionalidades de entrada, salida y procesado de audio.

[1] https://juce.com/

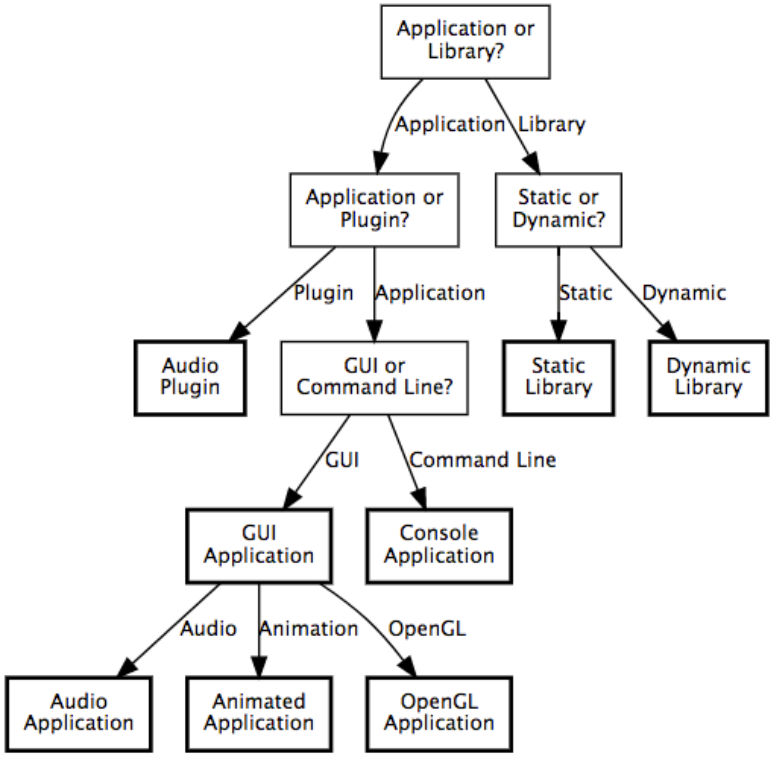

Fig. 3.2. – Esquema de relación entre los tipos de proyectos del framework Juce

Aplicación animada. También es muy similar a la aplicación GUI, pero en este caso se basa en clases orientadas a la creación de gráficos animados. Este tipo de proyecto se utiliza cuando se desean crear animaciones simples en la GUI de la aplicación.

Aplicación OpenGL. Este tipo de aplicación también es parecida a las aplicaciones GUI simples, pero sus clases están orientadas a la producción de gráficos en tiempo real. Se utilizará este tipo de proyecto cuando se quieran renderizar elementos gráficos complejos en la GUI de la aplicación desarrollada.

Aplicación de consola. Este tipo de proyecto se utiliza cunado no es necesario que la aplicación cuente con una interfaz de usuario.

Plugin de audio. Los proyectos de tipo plugin cuentan con una estructura totalmente diferente al resto de proyectos, trabajando con dos clases orientadas una de ellas a la parte gráfica de la aplicación y la otra a realizar el procesado de audio correspondiente. Este tipo de proyecto será utilizado cuando se quiera desarrollar un plugin que pueda ser alojado en un host, como podría ser cualquier plataforma DAW orientada a la edición de audio.

3.2.1. Plantilla de plugin de audio en Juce

La herramienta principal de Juce es el Projucer, una herramienta IDE que permite crear y gestionar los proyectos Juce de forma fácil utilizando una plantilla u otra según el tipo de aplicación que se quiera desarrollar. Las plantillas existentes son las que aparecen en la imagen 3.3..

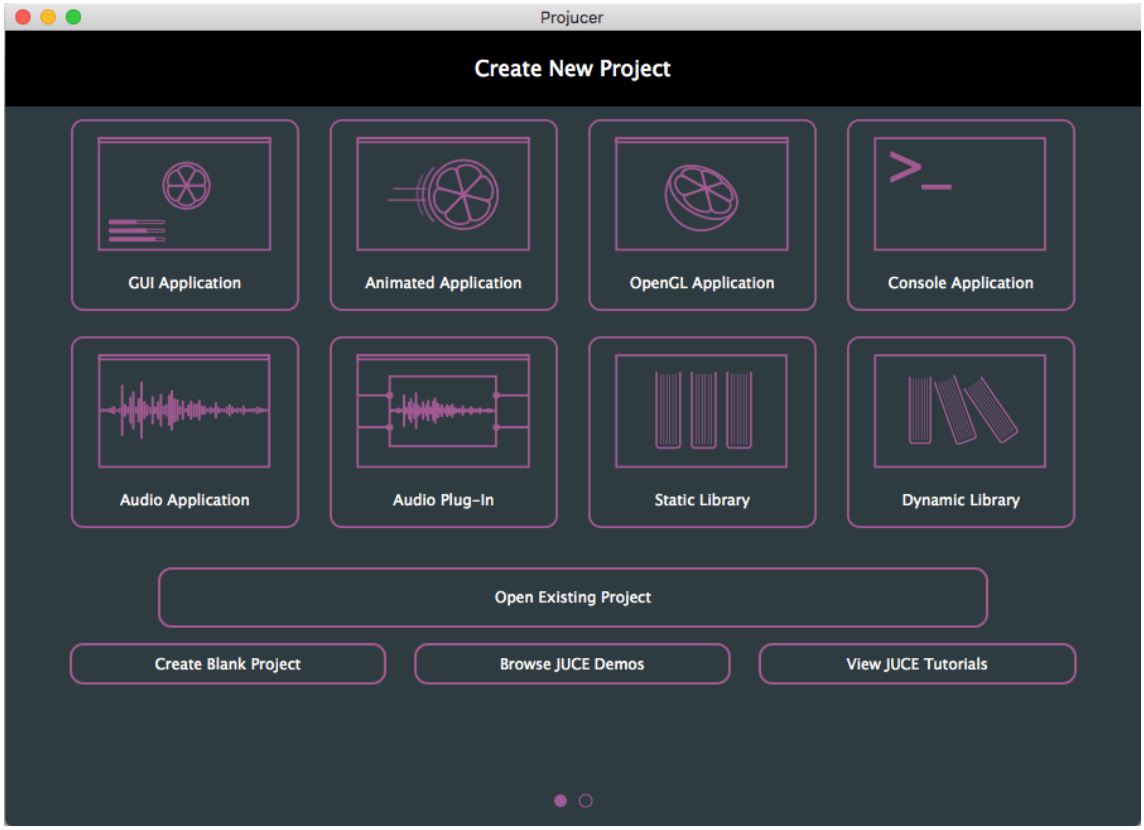

Fig. 3.3. – Ventana para la creación de un nuevo proyecto en Projucer

Cuando se crea un proyecto según una de sus plantillas genera una serie de archivos que permiten que el codigo se compile en las plataformas que se tienen como objetivo. Al utilizar la plantilla orientada a la programación de plugins de audio, JUCE utiliza dos clases principales.

AudioProcessor. es una clase abstracta que se dedica a realizar el procesado del audio, representa una instancia del plugin cargado en el host y es una wrapper class, envolviendo a los diferentes formatos de plugin VST, AU, RTAS y AXX. En las funciones de esta clase es donde se debe realizar la programación orientada al procesado que realizará el plugin. Los métodos o funciones mas importantes con las que cuenta son las siguientes.

AudioProcessor(). El constructor de esta clase cobra vital importancia en este proyecto, ya que en el se deben establecer el nuevo número de entradas y de salidas, al tratarse de un plugin dedicado a señales Ambisonic.

prepareToPlay(). Es llamada antes de que comience la reproducción para preparar correctamente el procesador de audio, desde ella es posible llamar a los métodos '*getTotalNumInputChannels()*' y '*getTotalNumOutputChannels()*' o consultar la variable miembro '*busLayout'* para averiguar la cantidad de canales que se deberán procesar. También permite acceder al valor '*maximumExpectedSamplesPerBlock'* que indica el número máximo de muestras de audio que se proporcionarán en cada bloque de procesado, este valor permite ajustar el tamaño de los buffers de audio que se utilizan de forma interna en el procesado.

processBlock(). Aunque los demás métodos también son necesarios, se podría decir que este es el mas importante de la clase '*AudioProcessor'* ya que realiza el procesado de cada uno de los

bloques de audio, dentro de el se debe realizar la programación del efecto que se desea realizar al audio. Para ello recibe como entrada un puntero al buffer de audio y otro al buffer Midi, para recibir también este tipo de mensajes y hacer la edición de audio en función de ellos. El buffer de entrada es el mismo que el de salida, por lo que esta función sobrescribe sustituyendo las muestras que contiene este buffer con los nuevas muestras

isBusesLayoutSupported(). Un método también importante en el caso de este proyecto, ya que comprueba si el número de canales del bus en el que se esta utilizando es el correcto, en el caso de utilizar señales Ambisonic de tercer orden como es este caso se deberá indicar dentro de este método que soporta esta configuración de bus.

getTotalInputChannels(). Este método devuelve el número total de canales de entrada, sumando el número de canales correspondiente a todos los buses que conforman las entradas del plugin de audio.

getTotalOutputChannels(). Este método devuelve el número total de canales de salida, sumando el número de canales correspondiente a todos los buses que conforman las salidas del plugin de audio.

getSampleRate(). Devuelve el valor correspondiente al número total de muestras por segundo que contiene el flujo de audio que se esta manejando.

getBlockSize(). Devuelve el tamaño, dado en número de muestras, de los bloques de audio que entran a través del buffer.

AudioProcessor::WrapperType. Además de los métodos comentados, la clase cuenta con una enumeración llamada '*WrapperType*' que contiene indicadores para establecer dentro de que contexto se está utilizando el plugin, los indicadores corresponden a los distintos formatos en los que podría trabajar el plugin desarrollado VST, VST3, Audio Unit, Audio Unit V3, RTAS, AAX, Standalone, Unity o Undefined.

AudioProcessorEditor. es una clase base de Juce destinada a contener las funcionalidades de la interfaz de usuario del plugin desarrollado. En esta clase se deberán instanciar los controles deseados para el plugin que posteriormente se enlazarán con variables de la clase '*AudioProcessor*'. Los métodos mas importantes correspondientes a esta clase son los explicados a continuación.

AudioProcesorEditor(). Es el constructor de la clase '*AudioProcessorEditor*' permite inicializar todas las variables y dar un estado inicial coherente a todos los campos de la clase.

Paint(). En el se programa como se deberá realizar el dibujado de los elementos gráficos de la interfaz gráfica de usuario

Resized(). Este método además de establecer si la ventana puede variar de tamaño o no, es un buen lugar para realizar la programación donde se indicará el tamaño que deben tener los elementos de la GUI y establecer su posición.

Al generar un proyecto con la plantilla para el desarrollo de plugins se generan cuatro archivos iniciales, que son con los que se deberá trabajar:

PluginProcessor.h y PluginProcessor.cpp son los archivos de cabecera e implementación del cuerpo de la clase '*PluginProcessor*'. Esta clase es derivada de '*AudioProcessor*', por lo que hereda sus constructores, métodos y campos, y será donde se realice la programación para el procesado de audio. En el archivo de cabecera '.h' se deben declarar los nuevos métodos y las variables miembro que se deseen añadir a la clase.

PluginEditor.h y PluginEditor.cpp son la cabecera e implementación del cuerpo de la clase PluginEditor. En este caso deriva de la clase '*AudioProcessorEditor*', heredando también sus constructores, métodos y campos. Aquí se realizará la programación orientada a la interfaz de usuario del plugin. Al igual que en '*PluginProcessor',* el archivo de cabecera '.h' se deben declarar los nuevos métodos y las variables miembro que se deseen añadir a la clase.

3.2.2. Host de plugins de Juce

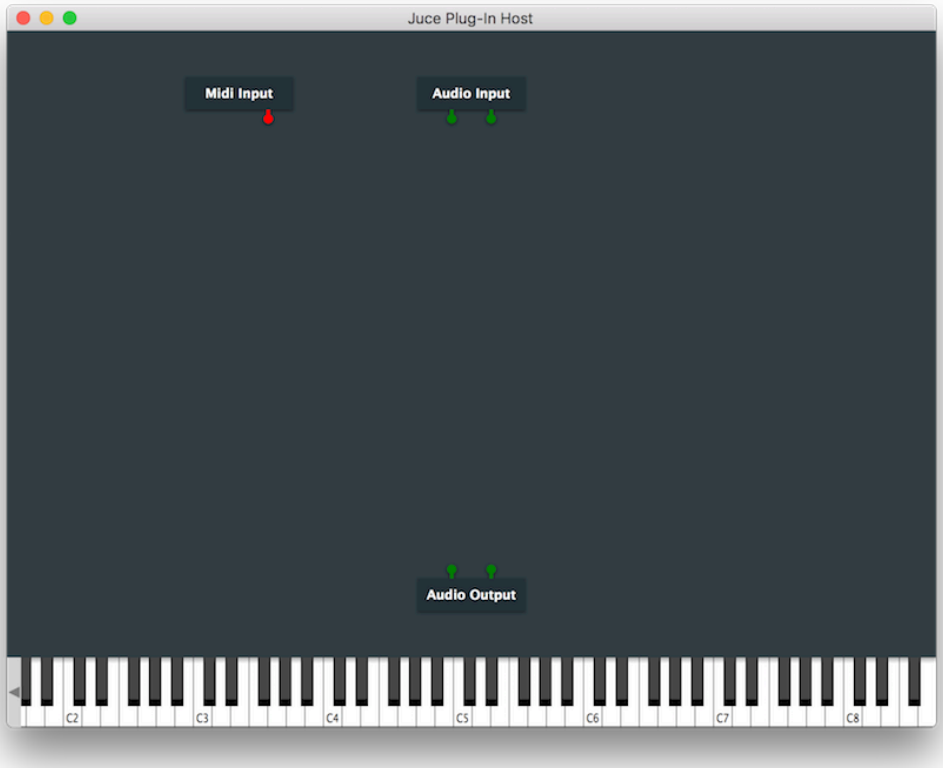

Fig. 3.4. – Host de plugins de Juce

Juce es ideal para el desarrollo de efectos de audio, ya cuenta con una plantilla dedicada a alojar plugins para una depuración mas sencilla, porque el software de este tipo no se puede ejecutar para realizar pruebas como si se tratase de una aplicación standalone.

Este host para plugins permite hacer las cadenas necesarias para las pruebas del software que se este desarrollando, pudiendo cargar tantos plugins como se desee. Con el se tienen tantas entradas y salidas de audio en función de lo que permita la interfaz que se este utilizando además de contar con entradas y salidas correspondientes a mensajes Midi.

Para probar un plugin de audio como es el caso de este proyecto, se podría realizar una cadena de este tipo que se expone como ejemplo. Primero se cargaría un plugin que permitiese generar audio, como podría ser un sampler, sintetizador o un simple reproductor de archivos de audio, como alternativa se podrían rutear las salidas de un DAW a las entradas del Host de Juce. El paso siguiente sería realizar la codificación a tercer orden Ambisonic, por lo que se insertaría en la cadena un plugin de codificación. Tras realizar la codificación de donde saldrían los 16 canales correspondientes al orden Ambisonic se colocaría el plugin que se esta desarrollando, que realiza el procesado de la señal Ambisonic. Después se realizaría la decodificación para bianuralizar la señal Ambisonic con otro plugin dedicado y sus salidas se rutearian a las salidas del host.

Esta o cualquier otra configuración puede ser guardada, y cada vez que se ejecute el software que esta siendo desarrollado desde la plataforma IDE de programación se llamará al host de plugins con la configuración ya establecida con lo que se pueden realizar pruebas rápidas o una depuración sencilla del software de plugins de audio en el que se esta trabajando.

3.2.3. Módulo DSP de Juce (Digital Signal Processing)

El modulo dsp de Juce [2] es una colección de diferentes clases orientadas principalmente a la implementación de filtros de distintos tipos, para la realización de este proyecto es conveniente conocer las clases que han sido utilizadas de este modulo de procesado de señales digitales.

dsp::StateVariableFilter. Esta clase se corresponde con un filtro IIR que puede cambiar de estado, pudiendo trabajar como filtro paso bajo, paso banda y paso alto. La pendiente de caída a partir de la frecuencia de corte es de 12 db/octava.

dsp::ProcessorDuplicator. Convierte una clase de procesado mono en una versión multicanal duplicándola y aplicando en todos los canales de un buffer de entrada de audio.

dsp::ProcessSpec. Permite establecer los ajustes necesarios que se necesitan para establecer un estado inicial coherente de los objetos pertenecientes al modulo dsp que se estén utilizando. Esta clase cuenta con tres campos básicos que corresponden a frecuencia de muestreo utilizada, máximo tamaño de los bloques que se van a procesar y número de canales a los que se va a aplicar el procesado.

dsp::StateVariableFilter::Parameters. *E*s una estructura utilizada para contener de forma organizada cada uno de los parámetros del filtro, como son frecuencia de corte, resonancia y tipo de filtro, paso bajo, paso banda o paso alto. Una vez establecidos sus valores correctamente se le puede pasar directamente al filtro toda la estructura correspondiente a los parámetros.

dsp::ProcessContextReplacing. Esta clase esta diseñada para su uso en situaciones en las que se usa un solo bloque de audio para el procesado, donde el bloque es el mismo para la entrada y la salida. Básicamente coge el bloque desde el buffer, se le aplica cierto procesado, en este caso el filtrado y tras esto reemplaza al bloque original que contenía el buffer.

[2] https://theaudioprogrammer.com/

dsp::AudioBlock. Permite dar al buffer original un nuevo alias que le permita trabajar como objeto perteneciente al modulo dsp.

4. DISEÑO E IMPLEMENTACIÓN DEL PLUGIN "MIRROR MAZE DELAY"

4.1. Estado del Arte

Actualmente existen plugins desarrollados por grupos de investigación para realizar el tratamiento de señales Ambisonic, es conveniente hacer un pequeño análisis de ellos para poder realizar la codificación y de codificación correctamente, además de conocer los procesos que es posible aplicar actualmente a las señales Ambisonic. A raíz de ellos se procederá al diseño de un nuevo software en forma de plugin de audio que permita realizar procesado de señales Ambisonic de una forma novedosa y no implementada hasta la fecha. Los siguientes plugins pertenecen a la casa IEM.

4.1.1 Suite de plugins IEM [3]

Fig. 4.1. – Logo de la suite de plugins IEM

AllRADecoder. Permite crear un decodificador Ambisonics para una distribución de altavoces hasta 7º orden. La codificación se aplica directamente al audio de entrada, por lo que se puede insertar en una pista dedicada a la decodificación en el proyecto.

BinauralDecoder. Procesa una entrada Ambisonics para adaptarla a un reproducción binaural con auriculares a través de HRIR.

CoordinateConverter. Convierte parámetros VST de representación esférica a representación cartesiana y viceversa, por lo que no realiza procesado de audio.

DirectionalCompressor. Compresor de señales Ambisonics que permite controlar el rango dinámico de regiones espaciales. Cuenta con controles para indicar la zona a la que se desea afectar. Tiene dos líneas de compresión que pueden afectar de forma omnidireccional, solo a la zona seleccionada o al resto de la señal, todo esto como añadido a los controles de un compresor clásico de audio.

DirectivityShaper. Permite filtrar el espectro en cuatro bandas y aplicar a cada una un patrón de directividad distinto, además de posicionarlas en la zona del espacio que interese.

[3] https://plugins.iem.at/

DistanceCompensator. Calcula el delay y ganancia aplicables para cada altavoz en función de la distancia a la posición de escucha.

DualDelay. Este plugin cuenta con dos líneas de delay que pueden ser configuradas de forma independiente con los controles posicionando el delay en la posición que interese del campo espacial.

EnergyVisualizer. Muestra la distribución de energía en la esfera de la señal de entrada el visualizador cuenta con el mapa de colores representados, situado junto a la zona proyección.

FdnReverb. Este plugin es una reverberación que actúa sobre señales de audio normales o Ambisonics.

MatrixMultiplier. Permite cargar una configuración que contenga un objeto de tipo TransformationMatrix que será aplicado a la señal de entrada.

MultibandCompressor. Divide la señal en 4 bandas que comprime por separado, para ello cuenta con los controles típicos de un compresor para cada banda.

MultiEncoder. Permite codificar varias fuentes a la vez, cada fuente puede ser paneada y tratada individualmente con los controles correspondientes.

MultiEQ. Cuenta con 6 filtros de ecualización que pueden cambiar entre varios tipos, encenderse o apagarse, el primero y ultimo pueden ser LP o HP.

OmniCompressor. Compresor que actúa sobre señales Ambisonics de manera omnidireccional, puede usarse como limitador.

ProbeDecoder. Permite decodificar la señal Ambisonics para una dirección de entrada especifica y escucharla.

RoomEncoder. Permite situar un oyente en una habitación virtual, de la que podemos establecer las dimensiones, en esta habitación pueden modelarse hasta 200 reflexiones. También se puede mover al oyente y la fuente, cambiar el carácter de las reflexiones, el coeficiente de reflexión y añadirse atenuación adicional a cada pared.

SceneRotator. Rota escenas Ambisonics, para ello permite voltear los ejes, los parámetros 'Yaw', 'Pitch' y 'Roll' son los encargados de realizar la rotación para cada sistema de coordenadas.

SimpleDecoder. Lee archivos JSON (Correspondientes a una configuración de altavoces generados con AllRADecoder) y a partir de la configuración introducida decodifica la señal de entrada Ambisonics para su reproducción en altavoces.

StereoEncoder. Codifica señales mono o estéreo a dominio Ambisonics y realizar su paneo de la forma deseada.

4.1.2. Suite de plugins Sparta [4]

AmbiBin. Decodificador binaural de Ambisonics hasta 7º orden, con soporte de tracking para cabeza, sirve para la reproducción en auriculares de señales Ambisonics (Format B) añadiendo un rotador. Permite importar HRIR propias y utilizar varios modos de decodificación.

AmbiDec. Decodificador Ambisonics dependiente de la frecuencia para altavoces. Permite hasta 64 canales y incluye presets 2D y 3D. Permite importar HRIR y se pueden elegir diferentes opciones de decodificador para baja y alta frecuencia.

AmbiDrc. Compresor de rango dinámico Ambisonics dependiente de la frecuencia, no cambia las propiedades espaciales aunque la percepción puede variar al comprimir la señal. Cuenta con los controles clásicos de un compresor ya explicados en el apartado dedicado al plugin DirectionalCompressor.

AmbiEnc. Codificador Ambisonics con entrada de hasta 64 canales que codifica como señal en Ambisonics para las direcciones especificadas. La resolución espacial esta determinada por el orden.

Array2SH. Codifica señales de matriz esférica/cilíndrica en señales armónicas esféricas (format B).

BeamFormer. Da forma al haz de emisión de una dirección concreta, con patrones posibles de forma Cardioide, Hipercardioide o MaxEV.

Binauraliser. Tiene entrada de hasta 64 canales. Realiza una convolución por HRTFs interpoladas aplicando ganancias VBAP y ITD. Incluye presets para 2D y 3D. También permite tracking mediante mensajes OSC y hacer rotación con controles 'Yaw', 'Pitch' y 'Roll'.

Dirass. Visualizador de campo sonoro que además, permite colocar secuencias de video detrás del mapa de actividad en tiempo real para crear una cámara acústica.

MatrixConv. Convolucionador de matrices. La matriz de filtros debe concatenarse y cargarse como un único archivo WAV.

Panner. Paneo 3D dependiente de la frecuencia basado en VBAP, cuenta con presets 2D y 3D, se pueden controlar las direcciones de cada canal entre entrada y salida.

PowerMap. Representa energía de sonido relativa o probabilidad de que una fuente llegue a una posición de escucha desde una orientación concreta, amarillo alta energía, azul baja. Permite visualizar señales de hasta 7º orden y reproducir video en tiempo real.

Rotator. Aplica una matriz de rotación Ambisonic seleccionando la rotación aplicada con los controles 'Yaw', 'Pitch' y 'Roll', se puede rotar con tracking de la cabeza a través de mensajes OSC.

[4] http://research.spa.aalto.fi/projects/sparta_vsts/

4.1.3. Suite de plugins Ambix [5]

Los siguientes plugins corresponden a la suite de Ambix para procesado de señales Ambisonic.

ambix_binaural. Se trata de un decodificador binaural que utiliza para la decodificación las respuestas al impulso de diferentes configuraciones de altavoces obtenidas de configuraciones físicas reales. La suite incluye varios presets correspondientes a diferentes configuraciones de altavoces para realizar la decodificación.

ambix_decoder. Trabaja como el decodificador binaural pero no realiza el paso de la convolución por las respuestas al impulso, las señales correspondientes a cada altavoz son enviadas directamente a las salidas establecidas de la interfaz de audio utilizada.

ambix_converter. Realiza conversión entre diferentes ordenes Ambisonic, permite trabajar con varios ordenes dentro de un mismo proyecto de producción de señales Ambisonic.

ambix_directional_loudness. Permite amplificar, atenuar o filtrar una región seleccionada del entorno.

ambix_encoder. Es un plugin de paneo tridimensional con diferente numero de canales de entrada, realiza la codificación de señales de otros tipos al ámbito Ambisonic. Cuenta con un parámetro 'Width' que permite distribuir los canales de forma uniforme sobre el azimut.

ambix_mirror. Invierte o realiza un espejado sobre cada uno de los ejes del sistema de coordenadas x, z e y.

ambix_rotator_z. Aplica transformaciones de rotación en torno al eje z a las señales Ambisonic de entrada.

ambix_rotator. Realiza rotaciones de la escena Sonora envolvente en torno a los tres ejes del sistema de coordenadas utilizado.

ambix_vmic. Se basa en el mismo principio que ambix_directional_loudness, pero no trabaja sobre una zona del campo sonoro establecida, si no sobre la señal recogida por un micrófono virtual para el muestreo de la escena sonora envolvente.

ambix_warp. Realiza un procesado de warping sobre el campo sonoro, comprimiendo la zona de los polos de la esfera de audio envolvente y expandiendo el ecuador o expandiendo los polos y comprimiendo el ecuador.

4.2. Metodología

Tras realizar este proceso de recopilación y análisis de la información necesaria para implementar software en forma de plugins de audio orientados a el procesado de señales Ambisonic, se procede a establecer los puntos clave que fijaran las características de este proyecto y la forma en que se llevará a cabo.

[5] http://www.matthiaskronlachner.com/?p=2015

4.2.1. Funcionalidades del plugin y parámetros a implementar

El primer punto a establecer es el uso que se va a dar de la teoría estudiada respecto a las señales Ambisonic, ya que en este trabajo se basa en el uso de este estándar de audio envolvente. Puesto que el efecto sonoro mas evidente y notorio que se puede realizar a través de transformaciones Ambisonic sencillas es la rotación azimutal de la escena sonora se ha decidido realizar una implementación de esta transformación en el software a desarrollar. Además la carga computacional que exige es mucho menor que en la realización de otros procesados por lo que aplicar este efecto en forma de plugin dentro de una producción de audio envolvente no generará una carga excesiva de trabajo a la estación de trabajo.

Se ha decidido también que el plugin sea uno de los efectos clásicos de audio pero también que actué trabajando de una forma creativa dentro del ámbito Ambisonics. Por ello se ha establecido que el efecto a implementar será un delay clásico, este efecto generará réplicas retardadas de las señal envolvente original y cada una de estas réplicas sufrirá una rotación angular dentro del plano de azimut. Con esto cada replica de la señal irá girando alrededor del punto de escucha con un ángulo elegido por el usuario del plugin.

Además como la mayoría de los efectos de delay de audio clásicos se podrá controlar el tiempo de separación entre cada réplica de la señal original y el numero de repeticiones que se producirán de esta. También para que el uso de este plugin de delay sea mas sencillo, y basando esto en los plugins de delay actuales, se ha decidido introducir un control que permita realizar una mezcla de la señal de entrada con la señal procesada. Con este control de mezcla entre las dos señales se podrá utilizar el plugin como efecto insertado en el propio bus por donde circula la señal Ambisonic o en un bus utilizado para el envió-retorno de la señal. De esta forma al utilizarlo como inserto se podrá hacer un efecto de delay mas o menos notorio a la señal utilizando este control y también al utilizar ese parámetro en su máximo valor, donde únicamente a su salida dará señal procesada y se hará el efecto mas o menos evidente controlando el volumen del bus de envío-retorno.

Por otro lado, para conseguir mayor versatilidad en el procesado, además de permitir ajustar la señal a las necesidades de la producción y darle un mayor realismo al efecto de delay, ya que muchos de los diseños de delay clásicos incluyen esta función, siendo casi un estándar en la producción de audio, se va a introducir en el diseño un filtrado de la señal procesada.

4.2.2. Diseño de la interfaz e imagen gráfica

En base a las decisiones tomadas en el apartado anterior, se han establecido como parámetros controlables del plugin de delay el tiempo para el retardo entre réplicas de la señal 'Time', el numero de nuevas replicas que aparecerán de la señal 'Feedback' y un control de 'Dry/Wet' para hacer el efecto mas o menos notorio. Este control de mezcla puede implicar una reducción o aumento considerable de la señal de salida del plugin se implementará un control de 'Output' que controlará el volumen general de salida para aumentarla o disminuirla en función de las necesidades de la producción.

Para la sección de procesado correspondiente a las transformaciones Ambisonic contará con un control 'Angle' para establecer la rotación que sufrirán las señales.

Por ultimo para la sección de filtrado se podrá seleccionar el tipo de filtro a utilizar consiguiendo así una gran versatilidad en los tipos de filtrados que se pueden realizar.

También se podrá seleccionar la frecuencia de corte y el factor Q de calidad del filtro utilizado.

La distribución de los controles dentro de la interfaz gráfica de usuario se realizará agrupando los controles en distintas secciones correspondientes a cada funcionalidad, quedando así distribuidas en una sección de filtrado, otra de control de los parámetros del delay, otra para el procesado Ambisonic y como última fase los controles correspondientes a la mezcla salida del plugin.

Por otro lado se ha realizado un planteamiento justificado de toda la imagen gráfica del efecto de audio. Para ello se ha diseñado la apariencia relacionándola con el entorno circular que se crea alrededor de la posición de escucha y la rotación que sufre la señal con un angulo fijado alrededor de esta, basando todo el fondo del diseño en un mandala, ya que estas imágenes se generan a partir de un patrón repetitivo circular, relacionándose directamente con el efecto de replicado y rotación que sufre la señal.

Por ultimo, como el efecto sufrido es capaz de generar una serie de espejados o repeticiones frente el eje de la posición de escucha, creando una señal de audio envolvente muy difusa y sin una posición fija formando una especie de laberinto, se ha realizado el diseño de un logotipo basado en este laberinto de expejados y llamando al plugin 'Mirror Maze Delay'.

Fig. 4.2. – Logotipo del plugin desarrollado

4.3. Desarrollo del proyecto

Una vez realizada la correspondiente recopilación de información y establecida la metodología que se va a utilizar a la hora de desarrollar este proyecto. La implementación se ha realizado de forma integra a través de programación realizada en el lenguaje C++ y ha sido soportada por el framework de Juce, que se basa en este lenguaje.

De esta forma se ha utilizado como punto inicial la plantilla de plugin de audio proporcionada por Juce y trabajando sobre ella se ha conseguido realizar todo el procesado de audio necesario, además del diseño de la interfaz gráfica de usuario. Se describen a continuación las funciones implementadas en las clases que proporciona esta plantilla, además de las modificaciones que se han introducido en las funciones ya proporcionadas por Juce.

Además de trabajar sobre las clases de Juce ha sido necesario diseñar un buffer circular y una clase para la inclusión de imágenes en la interfaz gráfica de usuario que se explican a continuación.

4.3.1. Diseño e implementación del buffer circular

Para la implementación del efecto de delay ha sido necesaria la creación de una estructura de datos especifica, como es un buffer circular, en este caso es útil para permitir acceder a datos correspondientes a muestras de audio atrás en el tiempo, es decir, que ya han sido reproducidas o procesadas.

El buffer circular es una estructura de datos que se va llenando y leyendo de forma cíclica, internamente es como si su salida y entrada estuviesen conectadas formando una estructura circular.

Para introducir los datos en el cuenta con un puntero de entrada, este va marcando la posición donde se deben ir escribiendo los datos que entran, comenzando por el punto inicial. Una vez alcanza el ultimo registro de almacenamiento correspondiente al buffer, su posición vuelve al primer registro y continua escribiendo a partir de el, sobrescribiendo la información que contenía sin borrar el resto, por lo que los datos siguen siendo accesibles.

Por otro lado cuenta con un puntero de salida, que indica el punto de donde se deben ir leyendo los datos. Este puntero actúa de la misma forma que el de entrada, comenzando a leer en el registro inicial del buffer y avanzando hacia adelante, una vez que llega al final se reinicia su posición y vuelve al punto inicial continuando a partir de ahí la lectura.

En este proyecto el buffer circular tiene su aplicación, como se ha dicho, en la lectura de datos que ya deberían haber sido reproducidos o procesados que realmente en la línea temporal habrían ocurrido antes, para ello su tamaño se ha planteado para que pueda acceder a datos de dos segundos atrás, estableciéndolo en numero de muestras como dos veces la frecuencia de muestreo, ya que este es el numero de muestras que aparecen cada segundo.

4.3.2. Clase AudioProcessor()

El proyecto se ha implementado trabajando sobre la plantilla básica de plugin de audio proporcionada por Juce, por lo que, se ha trabajado sobre algunas de las funciones que están implementadas por ella, añadiendo o modificando el código necesario para la implementación del procesado deseado dentro de la clase proporcionada '*AudioProcessor*'.

Ha sido necesario declarar una nueva variable para cada uno de los parámetros modificables en la interfaz grafica de usuario, estas variables serán actualizadas desde una función en la clase AudioProcessorEditor almacenará los valores de los sliders en ellas cada vez que suceda un cambio en los potenciómetros.

Por otro lado, se han creado dos nuevos buffers, uno para realizar una copia simple del buffer de entrada original '*mBufferCopy'*, ya que el buffer principal sufrirá todos los efectos del procesado y al final de la edición de audio es necesaria una copia de la señal original, y otro que actuará como buffer circular que será el que permita acceder a muestras de audio hasta dos segundos atrás en el tiempo '*mDelayBuffer*'.

AudioProcessor(). En el constructor de la clase que hereda de 'AudioProcessor' únicamente se ha establecido el numero de canales de entrada y salida del bus para que se ajuste a los requerimientos de Ambisonics de tercer orden, para ello se ha utilizado la función de la librería de Juce *'AudioChannelSet::ambisonic(3)'*. Estableciendo así las entradas y salidas a 16 canales.

prepareToPlay(). En la función de preparación previa a realizar el procesado del audio correspondiente los buses de entrada se establece el tamaño del buffer de delay que se ha añadido '*mDelayBuffer*', introduciendo el numero de canales correspondiente al tercer orden Ambisonic y el tamaño en muestras que debe tener.

Para establecer el tamaño en muestras que deberá tener hay que definir hasta que instante pasado de la señal de audio se desea acceder, en este caso se ha establecido que se pueda acceder hasta dos segundos atrás en el audio a partir de lo que esta entrando por el buffer principal, por lo que su tamaño en muestras será dos veces la frecuencia de muestreo, además se ha añadido un extra para agrandar un poco el tamaño y tener un poco de margen.

Tras esto se actualiza el valor de la variable que almacena la frecuencia de muestreo '*mSampleRate'* y se establecen las especificaciones necesarias de frecuencia de muestreo y tamaño del bloque de procesado del filtro, previamente a esta operación es reseteado para vaciarlo de muestras.

isBusesLayoutSupported(). En esta función, además del código preestablecido con el que cuenta la plantilla de plugin de Juce se ha añadido el código necesario para que el plugin soporte buses de 16 canales correspondientes al tercer orden Ambisonic.

processBlock(). Como ya se ha comentado previamente en este documento, este es el método mas importante para el procesado de audio dentro del entorno de desarrollo de plugins en Juce. En el se ha añadido el código para realizar todos las ediciones necesarias al audio de entrada. Como primer paso se realiza una copia del buffer original que servirá al final de todo el procesado para generar el funcionamiento del control 'Dry/Wet', mezclando esta copia de la señal original con la señal de audio procesada en la proporción deseada. Tras esto, se llama a una de las funciones añadidas, llamada *'fillDelayBuffer()'*, que se encarga de ir llenando con muestras cogidas de las entradas de audio del plugin el buffer circular que se utiliza como buffer de delay. Una vez introducidas las muestras en el, se le pide que retorne las muestras introduciéndoles un retardo temporal, de esto se encarga la función '*getFromDelayBuffer()'*.

En este momento ya se cuenta con la señal retardada, por lo que es el momento de aplicarle a ella el procesado Ambisonic que introduce la rotación sobre el eje z. Tras esto se llama al método '*feedbackDelay()'* que se encarga de replicar la señal retardada aplicándole una ganancia que va haciendo que disminuya su amplitud con cada réplica. Por ultimo se le aplica a toda la señal retardada y a sus réplicas el filtrado correspondiente y esta señal es mezclada según el valor del control 'Dry/Wet' con la copia de la señal original que se había realizado al principio del proceso.

Además de trabajar sobre las funciones ya implementadas por la plantilla de plugin de audio de Juce, se ha creído conveniente realizar nuevas funciones dentro de la clase de procesado para una mayor organización, eficiencia y simplificación del código, con lo que se consigue que sea mas sencillo de entender separando los diferentes procesos que se realizan en bloques. Los métodos que se han creado son los siguientes.

updateFilter(). Este método se encarga de actualizar los valores correspondientes al filtrado, para ello se toman los valores de las variables que almacenan el valor del parámetro correspondiente a los controles de 'Filter Type', 'Cutoff' y 'Resonance'. Según el valor en 'Filter Type' de forma condicional se establece el estado del filtro como paso bajo, paso alto o paso banda y dentro de esta estructura condicional, tras establecer el estado del filtro, se ajusta el valor de los parámetros de frecuencia de corte y factor de calidad Q en función de los valores obtenidos en los potenciómetros de 'Cutoff' y 'Resonance'.

Rotation(). Dentro de este método se realiza la transformación de la señal Ambisonic según la matriz de rotación sobre el eje z explicada en el apartado correspondiente a este efecto. En lugar de utilizar operaciones matriciales, el procesado se ha simplificado utilizando operaciones simples entre canales, para ello se trabaja a nivel de muestra. Como ejemplo se coge la primera muestra de cada canal y se realizan las operaciones correspondientes entre ellas, tras esto se sustituye la muestra original en el buffer por la nueva muestra obtenida tras realizar el procesado pasara a la siguiente muestra y así sucesivamente, hasta completar todas las muestras del buffer para cada canal.

fillDealyBuffer(). Esta función se encarga de rellenar con muestras el buffer creado para la señal de retardo '*mDelayBuffer*' para ello se procede de la forma explicada para el buffer circular, es decir, cuando se llene por completo el buffer, el puntero de escritura volverá al punto inicial de este y continuara rellenando de muestras ahí sobrescribiendo cada una de las viejas muestras y sin vaciar el resto del buffer para poder seguir accediendo a la información que contiene.

getFromDelayBuffer(). Este método devuelve bloques de audio procedentes del buffer de delay que han sufrido el retardo deseado, para ello hace una copia de muestras anteriores a las que se están reproduciendo en el momento de ser llamado. Trabaja con el buffer circular antes mencionado, por lo que la copia se va realizando también de forma circular.

feedbackDelay(). Este método se encarga de sumar al buffer de delay copias retardadas de la señal procesada. A cada una de estas copias se le aplica una ganancia menor a 1, dada por el potenciómetro de 'Feedback', por lo que las replicas van disminuyendo en amplitud conforme la copia de ellas avanza en el tiempo.

Se ha creído mas conveniente a la inclusión de un diagrama de funciones, utilizar un esquema correspondiente al procesado que sufre el audio dentro de la función '*processBlock()*', ya que cada proceso es fácil de relacionar con las funciones que han sido explicadas, y el resto de relaciones entre las funciones dependen mas de la propia estructura interna de Juce que de la implementación que se ha realizado.

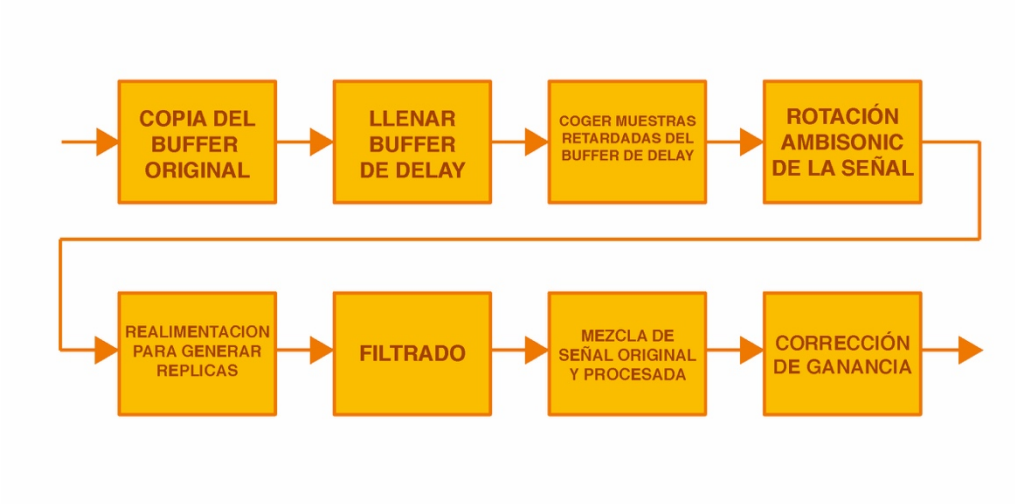

Fig. 4.3. – Esquema del procesado de audio

4.3.3. Clase AudioProcessorEditor()

En el archivo de cabecera de la clase 'AudioProcessorEditor' ha sido necesario declarar un conjunto de objetos de la clase Slider, que se utilizarán como controles rotatorios dentro de la interfaz gráfica de usuario. Además, también se han tenido que declarar un listener para detectar los cambios que se producen en los sliders y un campo del tipo '*Image*', que será necesario para dibujar el fondo.

AudioProcessorEditor(). Constructor de la clase, en el se establece el tamaño del editor GUI, por lo que se ha ajustado el tamaño según lo deseado, además se ha introducido el código necesario para darle unas propiedades iniciales a cada uno de los controles rotatorios. De ellos se establece el tipo de slider, el rango, valor inicial, color…; Por ultimo, cada uno de ellos se añade al editor y se establece como visible y se les añade el listener instanciado para que sea llamado cuando uno de los controles sufra un cambio de valor.

Paint(). En el se ha introducido el código correspondiente para que se pinte la imagen de fondo de la interfaz gráfica de usuario, donde se muestran los nombres de los potenciómetros y su rango de valores, además del logo y los gráficos decorativos del plugin. Para ello se ha utilizado el método '*Graphics::drawImage()*' que pinta en el panel objetos de tipo imagen.

resized(). En este método predefinido por la plantilla de plugin de audio de Juce se ha añadido el código correspondiente a definir el tamaño de cada uno de los controles rotatorios con los que cuenta el plugin además de su posición. Para conseguir esto se llama al método del componente de la GUI correspondiente '*Component::setBounds()*' al que se le pasan los valores de ancho, alto, posición *x* y posición *y*.

Además de los métodos proporcionados por la plantilla de plugin de audio de Juce, ha sido conveniente la creación de un nuevo método para realizar un procesado específico de el software desarrollado.

sliderValueChanged(). Se ha creado un método añadido a la plantilla de Juce, al que se llama cada vez que cambia el valor de uno de los componentes de la interfaz gráfica de usuario. Dentro de este método, se le asigna el valor de cada uno de los parámetros variables,

a las variables que recogen estos valores en la clase AudioProcessor para ello se utiliza el método '*Slider::getValue()*' con el que se puede obtener el valor de cada uno de los controles rotatorios.

4.3.4. Clase Images()

Para la inclusión correcta de imágenes en software desarrollado en el entorno de Juce, es conveniente la creación de una clase llamada Images, que se encarga de enlazar cada uno de los archivos de imagen con el proyecto en el que se esta trabajando. Permite modificar añadir o eliminar imágenes fácilmente al programa desde Projucer, sin tener que añadir nuevo código, ya que tras enlazar una imagen, Projucer genera automáticamente la clase Images con el código de enlace a ellas.

4.4. Funcionamiento

Previamente a describir el funcionamiento y modo de uso del plugin desarrollado, es conveniente conocer como proceder para configurar un proyecto dentro de un DAW para poder trabajar con él y que su funcionamiento sea el adecuado. En este caso se utilizara como ejemplo Reaper, ya que es un software de uso gratuito y cumple con los requerimientos necesarios para trabajar con señales Ambisonic.

A la hora de elegir el software para realizar la producción de audio en el ámbito Ambisonics es necesario asegurarse de que cumple con el principal requerimiento, que es soportar el trabajo con buses multicanal con un número de canales suficiente respecto al orden en el que se quiera trabajar. En este caso, el plugin trabaja en el tercer orden Ambisonics, por lo que se necesitaran buses de 16 canales.

4.4.1. Configuración del proyecto para producción de señales Ambisonic en Reaper

Para trabajar en proyectos orientados a reproducirse en un sistema High Order Ambisonics es recomendable trabajar con una plantilla de este tipo, ya que permitirá por un lado tener un ruteo con las correspondientes salidas físicas al sistema de altavoces y realizar una monitorización realista a través de auriculares, lo que dará una idea del sonido que se obtendrá en el punto de escucha real de la configuración de altavoces.

El esquema será el siguiente, contando con un bus principal dedicado a la señal Ambisonics y dos buses de salida separados para la reproducción en auriculares y altavoces. Por otro lado se tendrán que insertar tantas pistas individuales de audio como fuentes se vayan a utilizar, estas fuentes podrán ser señales mono, estéreo o Ambisonic, ya que estos tres tipos de señales son las que pueden ser codificadas o convertidas de orden para ser introducidas posteriormente en el bus Ambisonic. Al final de este bus se obtendrá una señal de audio envolvente que será enrutada hacia el bus de altavoces donde sufrirá la decodificación para adaptarla a la configuración física de reproducción. Una copia de esta señal Ambisonic sera enrutada hacia el bus de auriculares donde será decodificada para convertirse en una señal binaural.

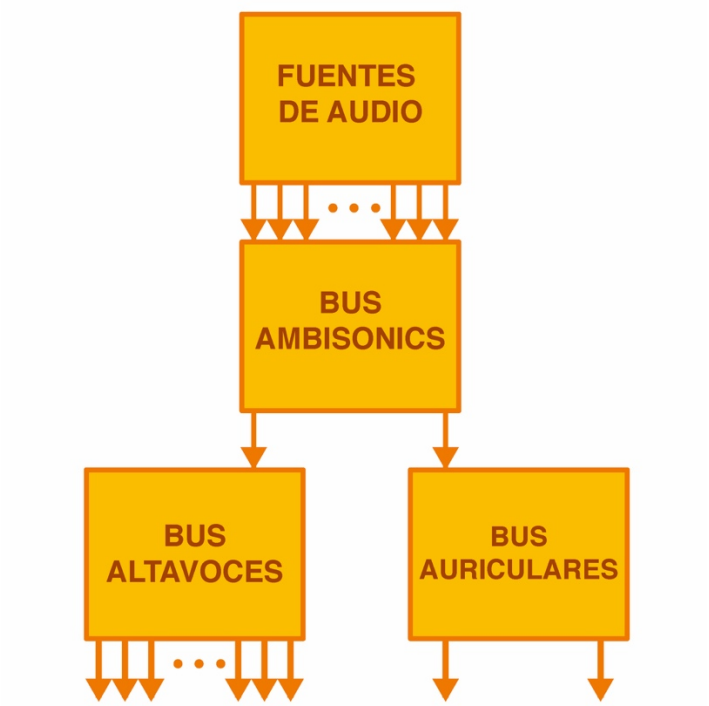

Fig. 4.4. – Esquema de ruteos dentro de la plantilla para producciones Ambisonic

El primer paso para implementar este esquema es la creación de los tres buses, para ello se creará un nuevo proyecto en Reaper y dentro de el tres buses que serán nombrados para su correcta identificación como 'Ambisonics', 'Altavoces' y 'Auriculares'. Para ello tras crear el nuevo proyecto, se deberá hacer click derecho sobre la zona reservada para los buses y se seleccionará la opción 'Insert new track'.

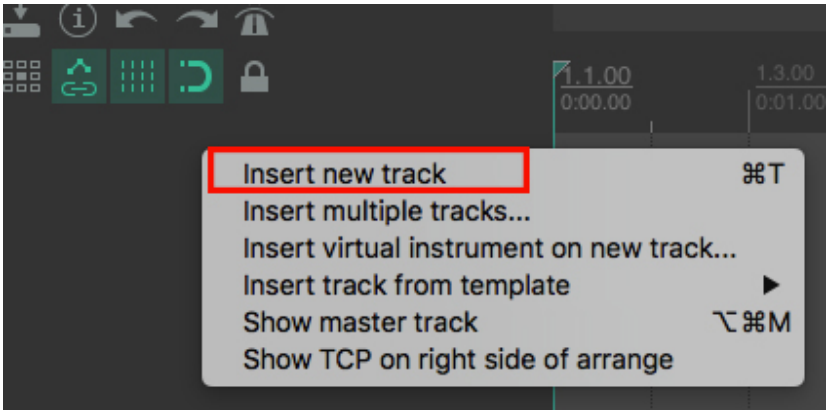

Fig. 4.5. – Creación de un nuevo bus

Por defecto en Reaper estos buses son estéreo y se encuentran enrutados al bus Master, por lo que el siguiente paso sera, en cada uno de los tres buses creados y también para nuevas pistas correspondientes a fuentes de sonido, pulsar el botón 'Route', que se encuentra enmarcado en rojo en la siguiente figura, del bus y desactivar la casilla 'Master send', enmarcada en verde. Si se desea realizar la monitorización a través de auriculares, la casilla 'Master send' se dejará activada para el bus 'Auriculares'.

También dentro de la ventana de configuración que aparece cuando se pulsa el botón 'Route' sera necesario ajustar el numero de canales según el orden Ambisonic a utilizar, en este caso 16 canales. Para ello cambiamos el valor del selector 'Track channels', enmarcado en azul, para darle el valor de 16.

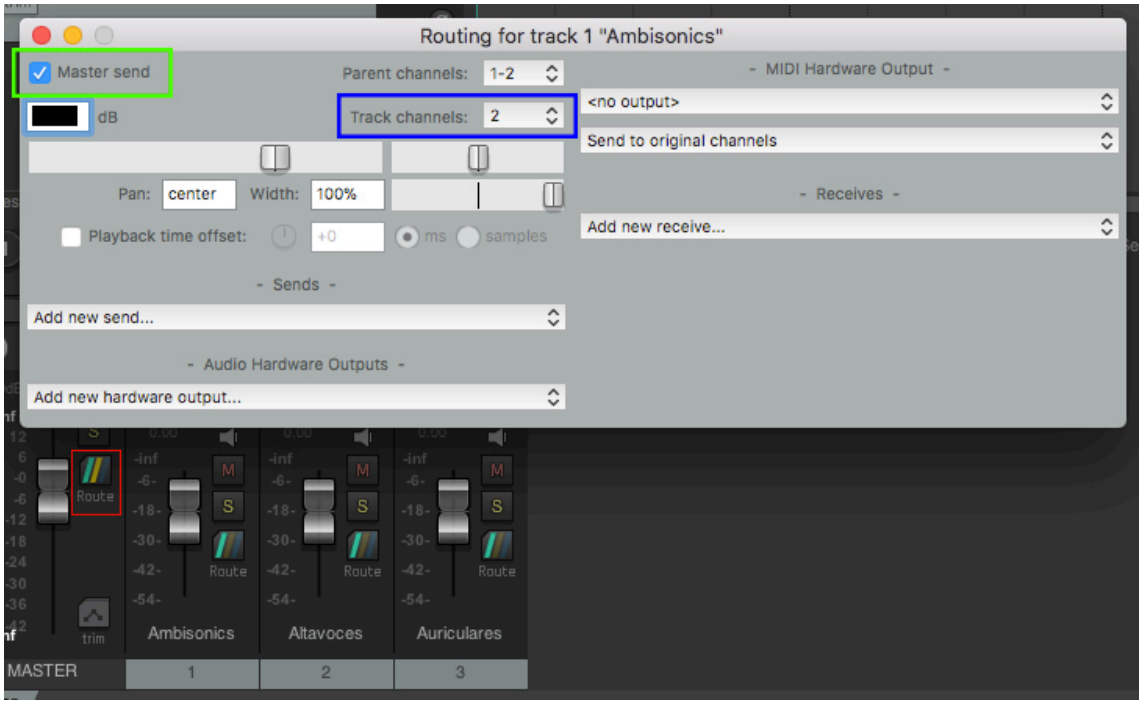

Fig. 4.6. – Ajuste en la ventana de ruteo

El segundo paso para implementar esta plantilla seria realizar una correcta conexión de los buses creados, para ello basta con arrastrar el botón 'Route' de un bus al del bus al que interese conectarlo. Primero se realizara este proceso ruteando el bus 'Ambisonics' hacia el bus 'Altavoces'. Tras hacer esta conexión aparecerá la ventana correspondiente al enrutamiento, en ella se deberá seleccionar la fuente multicanal de 1-16 para que todos los canales del bus se dirijan hacia este nuevo destino. Una vez se haya realizado correctamente el ruteo habrá que proceder a conectar el bus 'Ambisonics' al bus 'Auriculares' de la misma manera. En la siguiente imagen se encuentra marcado en rojo el selector que permite elegir le fuente multicanal en la ventana de ruteo.

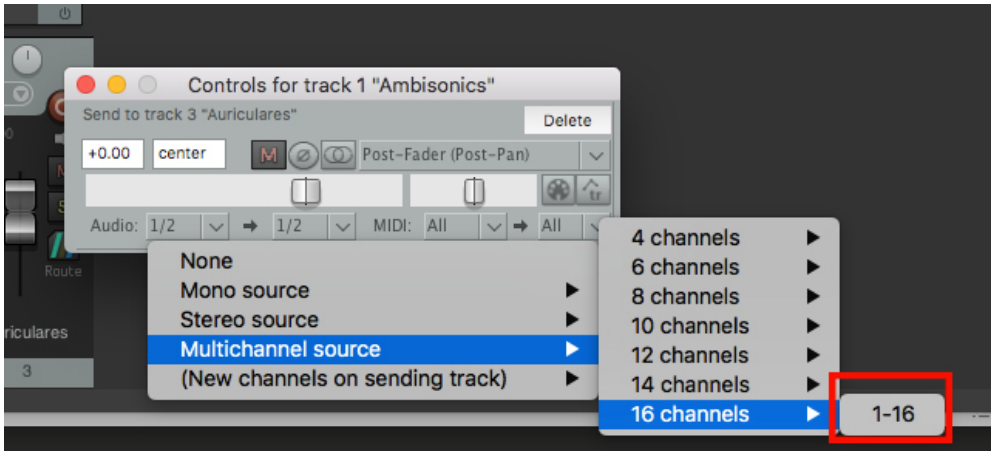

Fig. 4.7. – Ajuste de la interconexión de buses

Una vez realizado el ruteo se deberán cargar en la sección de efectos de cada bus los plugins necesarios para realizar la codificación, la decodificación y además obtener una representación gráfica de lo que esta ocurriendo.

En el bus 'Ambisonic' se cargara el plugin 'EnergyVisualizer' de IEM, este es el que dará una representación gráfica del campo sonoro envolvente.

En el bus 'Altavoces' se insertará en primer lugar el plugin 'SimpleDecoder' de IEM que permitirá cargar una archivo correspondiente a una configuración de altavoces preestablecida que se podrá generar con el plugin 'AllRADecoder'. En el caso de tratarse de una configuración de altavoces bidimensional no podremos usar estos plugins, y se debería utilizar 'sparta_ambiDEC', donde se podrá realizar directamente la configuración del sistema de altavoces, este plugin también se puede usar directamente en configuraciones tridimensionales como alternativa a 'SimpleDecoder'.

Por último en el bus 'Auriculares' se cargará el plugin 'BinauralDecoder' de IEM, que se encargara de realizar una adaptación de la señal para ser reproducida en auriculares manteniendo una simulación de las características envolventes de ella.

En la siguiente figura, 4.8. se muestra en rojo como acceder a la ventana que permite insertar efectos y, en verde, como añadir un nuevo efecto.

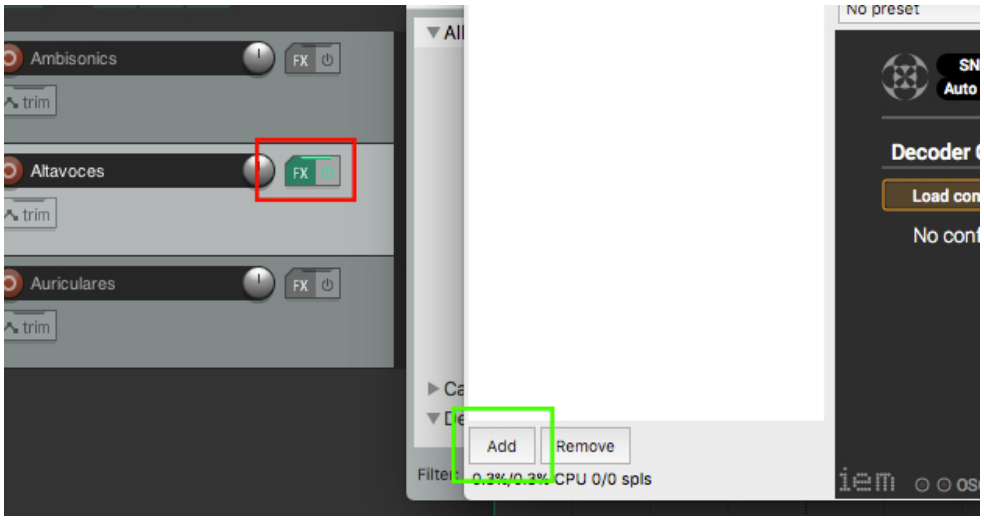

Fig. 4.8. – Añadir un nuevo plugin

Como siguiente paso habrá que añadir las salidas hardware correspondientes al interfaz de audio que se utilizará para el bus 'Altavoces', para ello se pulsara de nuevo en el botón 'Route' del bus y en la zona dedicada a la configuración de hardware, marcada en rojo en la siguiente figura, se seleccionará la primera salida a utilizar de la interfaz de audio. Para realizar un envió a la tarjeta de sonido de tantos canales como altavoces se estén utilizando se hará click en la zona marcada en verde en la figura 4.9. para cargar una fuente multicanal con el número de altavoces deseado.

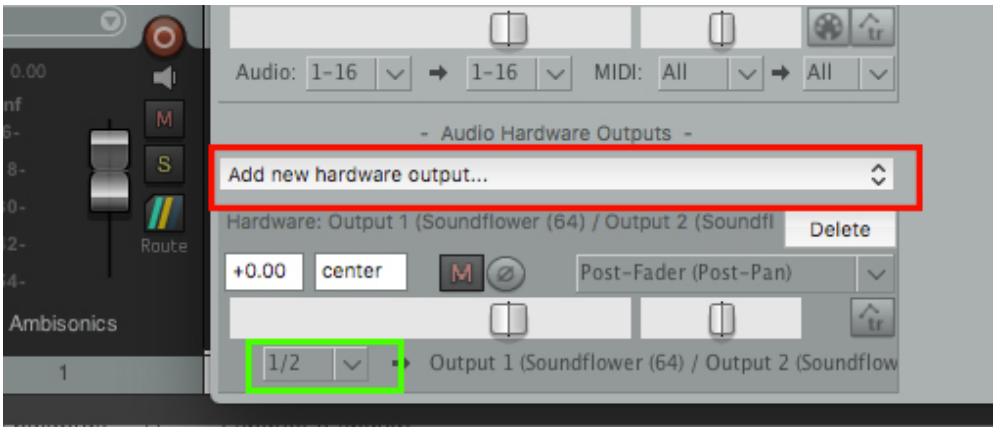

Fig. 4.9. – Establecer salidas físicas hacia la configuración de altavoces

Tras realizar estos pasos la plantilla estará preparada para realizar una producción Ambisonic, con capacidad para monitorizar el resultado a través de altavoces o auriculares. El siguiente y ultimo paso seria añadir las pistas de fuente de audio, donde se cargará un plugin capaz de realizar la codificación Ambisonic en caso de ser señales monofónicas o estereofónicas, una buena herramienta para realizar este proceso es el plugin 'StereoEncoder' de IEM tras esto se arrastrará el botón 'Route' de cada pista hacia el bus 'Ambisonics' y se ajustara el ruteo para que se envíen todos los canales del bus.

Es recomendable colocar el plugin 'EnergyVisualizer' en la cadena de efectos tras realizar la codificación para ver que se está realizando de la forma deseada.

4.4.2. Funcionamiento del plugin Mirror Maze Delay

Mirror Maze Delay es un efecto implementado para tercer orden Ambisonic, por lo que cuenta con 16 canales de entrada y 16 canales de salida, se ha elegido este orden por que será el utilizado en el proyecto Sonorizarte y la intención es la aplicación directa de este plugin en las producciones realizadas en este proyecto.

El plugin realiza un procesado de delay clásico, obteniendo a su salida la señal retardada y repeticiones temporales de esta. Como añadido a este efecto las repeticiones de la señal sufren un filtrado seleccionable y una rotación alrededor del eje vertical situado en el punto de escucha a través de transformaciones con matrices Ambisonic.

Se recomienda utilizar una plantilla de DAW como la descrita en el apartado anterior para el uso del plugin 'Mirror Maze Delay', aunque se puede utilizar dentro de configuraciones similares.

El plugin desarrollado permite usarse de dos maneras, como efecto insertado en el bus, o como efecto de envío-retorno en un nuevo bus.

Para usarlo como inserto se deberá cargar el plugin en el bus por el que discurre la señal Ambisonic que se quiere procesar, en el caso de utilizar la plantilla será el bus llamado 'Ambisonics'.

En el caso de querer usarlo como envió retorno se deberá crear un nuevo bus con el mismo numero de canales que el bus 'Ambisonic' y se insertará en el, tras esto se enviara la señal

que circula por el bus 'Ambisonic' a este nuevo bus y luego se retornará a el, para ello, en el caso de estar trabajando con el DAW Reaper, primero se arrastra el botón 'Route' del bus 'Ambisonics' al nuevo bus, y luego de este nuevo bus al bus 'Ambisonics'. En ambos casos habrá que ajustar el numero de canales que se quieren enviar en la ventana de ruteo.

Una vez elegido el modo de uso y ya insertado el plugin se podrá realizar el procesado deseado. Para ello 'Mirror Maze Delay' cuenta con los controles que se explican a continuación.

Fig. 4.10. – Interfaz gráfica de usuario del plugin desarrollado

Filter Type. El plugin cuenta con una sección dedicada al filtrado, este filtrado afectará a toda la señal de forma independiente a la posición de la fuente, de la señal, con un filtro de estado variable, que permite seleccionar entre tres tipos de filtro, este selector nos permite hacer el cambio entre ellos. Las opciones son LP, lo que realizara un filtrado paso bajo dejando pasar únicamente los graves de la señal, BP, que realizara un filtrado paso banda seleccionando la banda deseada y HP que realizará un filtrado paso alto dejando pasar las frecuencias mas altas de la señal envolvente.

Cutoff. Este control también corresponde al filtrado de la señal y permite seleccionar la frecuencia de corte del filtro, su utilidad cambia en función del tipo de filtro. En el caso de filtrado paso bajo 'LP' y paso alto 'HP' establece la frecuencia de corte y en el caso de utilizar la opción de filtro paso banda 'BP' establece el centro de la banda de paso. Los valores de este parámetro pueden ir desde 20 Hz a 20000 Hz, por lo que en el caso de no querer utilizar ningún filtrado se podrá seleccionar el filtro paso bajo y establecer el 'Cutoff' a 20000 Hz o seleccionar el filtro paso alto y configurar un 'Cutoff' en 20 Hz.

Resonance. El ultimo parámetro dedicado a la sección de filtrado corresponde a un control para establecer el factor de calidad 'Q' del filtro, al aumentar el valor de este potenciómetro se creará un pico de resonancia a la frecuencia de corte del filtro, lo que permitirá realizar modificaciones creativas a la señal que esta siendo procesada. Este parámetro tiene un rango de actuación con valores que van desde 0 a 5.

Delay Time. Este potenciómetro permite ajustar el tiempo de retardo o 'delay' que sufre la señal procesada con respecto a la señal de entrada, permite retrasar la señal entre 50 ms y 500 ms.

Feedback. El parámetro correspondiente al 'Feedback' permite hacer que la señal retardada sufra una retroalimentación y se vayan añadiendo a ella nuevos retrasos de la señal con el mismo tiempo que el seleccionado en el control 'Delay Time'. En su valor mínimo donde no introduce retroalimentación solo se escuchara la señal retardada en función del tiempo de retardo seleccionado. Al comenzar a aumentar su valor se comenzaran a escuchar nuevas replicas de esta señal cada vez mas retardadas temporalmente en múltiplos enteros del tiempo seleccionado en 'Delay Time'. Este parámetro permite ser establecido en valores entre 0 % y 95 %. Cuando se sitúa en su valor máximo hay que tener cuidado, dependiendo con que señal se esté trabajando, ya que la retroalimentación puede producir un aumento exponencial de la señal de salida con lo que saturará el canal.

Angle. Este potenciómetro permite realizar una rotación de la señal que esta siendo procesada sobre el eje *z*, su valor determina en cuantos grados girará la señal retardada. En el caso de utilizar el 'Feedback' en su valor mínimo solo rotará la señal retardada, en el caso de aumentar su valor y obtener repeticiones temporales de la señal, cada una de estas repeticiones sufrirá esta rotación, haciendo que cada una de ellas aparezca en el múltiplo correspondiente del ángulo inicial. En este caso las repeticiones de la señal irán rotando alrededor del punto de escucha obteniendo así el efecto de señal envolvente que aplica este plugin. Su valor permite ir desde -180 grados a 180 grados, consiguiendo así una circunferencia completa alrededor del punto de escucha.

Dry/Wet. Este control es el que hace que el efecto pueda ser utilizado como inserto en el canal y como efecto de envió-retorno. En el caso de utilizarse como inserto permite 'manchar' mas o menos la señal original, es decir, según su valor mezclara proporcionalmente la señal original de entrada en el plugin con la señal procesada que ha sido retardada, filtrada y rotada. Por ejemplo, si su valor se encuentra en 40 % la señal de salida será suma será la suma de un 40 % señal de entrada sin procesar y un 60 % señal procesada. En su valor 0 % haría de 'bypass' permitiendo pasar la señal original sin aplicarle ningún procesado, este control es útil para hacer mas o menos notorio el efecto en función de los intereses del productor.

Por otro lado si se prefiere utilizar el procesado clásico utilizado en los efectos de 'delay' de insertarse en un bus de envío-retorno se puede posicionar el control 'Dry/Wet' en su valor máximo de 100 %. En este caso a la salida del plugin únicamente se obtendrá la señal procesada y la mezcla con la señal original se realizara con el potenciómetro de volumen correspondiente al canal de envío retorno.

Output. Por ultimo para un ajuste mas cómodo del volumen de salida, ya que el procesado puede introducir un aumento o disminución de la ganancia, el plugin cuenta con un potenciómetro dedicado a aplicar una ganancia a la señal final. Sus valores van de 0 a 1.2, permitiendo aumentar o disminuir la señal final.

4.5. Análisis de los resultados

Para realizar una prueba cuantificable de que los resultados obtenidos con el procesado son los deseados, se han utilizado varios plugins de medición, todos ellos de uso libre y descarga

gratuita, que permiten ver gráficamente que el procesado es correcto y que el plugin funciona correctamente.

4.5.1. Filtrado

Para realizar un análisis del filtrado se ha utilizado un plugin llamado 'Span', de la compañía Voxengo, que trabaja como analizador de espectro y esta muy estandarizado en la industria de producciones de audio, ya que hace una representación muy precisa y de gran calidad siendo además un software gratuito.

Para realizar las pruebas correspondientes al filtrado se ha utilizado una señal de ruido blanco, para tener información de audio en todo el espectro, y así ver la banda frecuecial que desaparece al aplicar este proceso. La respuesta en frecuencia original de la señal se puede apreciar en la siguiente imagen 4.11..

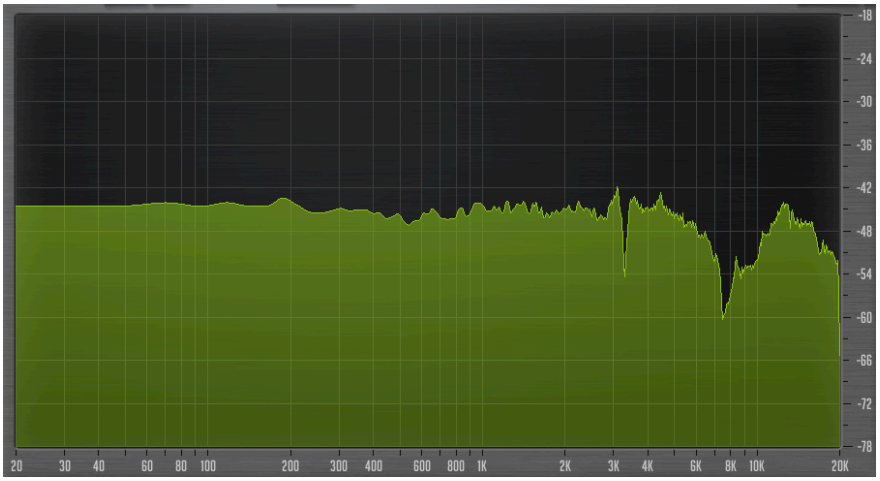

Fig. 4.11. – Señal original de ruido blanco

En base a esto, la primera y mas sencilla prueba que se ha realizado ha sido la del control correspondiente al 'Dry/Wet', para comprobar que efectivamente cuando su valor esta establecido como 0 % la señal pasa por el plugin sin sufrir ningún tipo de modificación espectral. Obteniendo como resultado la imagen 4.12-.que aparece a continuación.

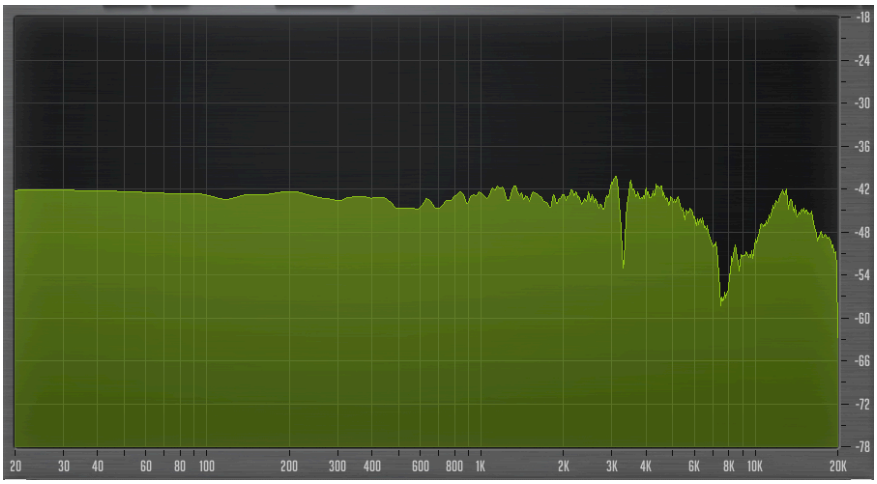

Fig. 4.12. – Señal de ruido blanco pasada por el plugin sin procesado

En ella se aprecia claramente que la función del plugin de dejar pasar completamente la señal original funciona a la perfección, ya que la grafica mantiene la forma original, y los pequeños cambios que se aprecian en ella, respecto a la imagen anterior, son debidos al instante temporal que se esta reproduciendo de la señal de ruido blanco y no al procesado que realiza el plugin.

Tras esto se ha realizado un procesado de filtrado paso bajo utilizando el plugin obteniendo a su salida únicamente la señal procesada, para ello el control de 'Dry/Wet' se ha establecido a su valor máximo de 100 %. El selector 'Filter Type' se ha fijado en su opción LP para que el filtrado sea paso bajo y el potenciómetro 'Cutoff' correspondiente a la frecuencia de corte se ha fijado en 1000 Hz utilizando este valor de frecuencia como referencia en todas las mediciones realizadas, por último el valor para el control de 'Resonance' se ha establecido a 1, donde no aplica realce a la frecuencia de corte. El resultado se puede apreciar en la siguiente gráfica 4.13..

Fig. 4.13. – Señal de ruido blanco con filtrado paso bajo

Como se aprecia en la figura 4.13. la señal se mantiene intacta por frecuencias inferiores a 1000 Hz, en esta frecuencia no aparece realce y por encima de ella la señal decae como debería, con lo que el procesado es correcto.

A continuación se ha realizado una nueva medición manteniendo los valores de los parámetros de tipo de filtro y frecuencia de corte y aumentando el control de 'Resonance' a su valor máximo, aumentando así el factor de calidad Q del filtro al valor máximo que permite el plugin, en la siguiente gráfica se puede ver el resultado obtenido.

Fig. 4.14. – Señal de ruido blanco con filtrado paso bajo y aumento del factor Q

Se ve que se mantiene el filtrado, pero en este caso la frecuencia de corte, 1000 Hz sufre un realce de 12 db, con lo que el resultado es el deseado.

Para la siguiente medición realizada se ha vuelto a utilizar un valor de frecuencia de corte de 1000 Hz y un valor de resonancia de 1. En este caso se ha cambiado el estado del filtro para que actúe como filtro paso alto eliminando la banda correspondiente a baja frecuencia. Se ha obtenido como resultado la gráfica siguiente 4.15..

Fig. 4.15. – Señal de ruido blanco con filtrado paso alto

En la gráfica se aprecia que, de nuevo, el resultado obtenido es el deseado, eliminando la banda situada por debajo de la frecuencia de corte de la forma correcta y dejando intacta la banda superior a ella.

Como siguiente paso, al igual que en filtrado paso bajo, se han mantenido los valores de tipo de filtro y frecuencia de corte utilizados y se ha aumentado el valor de resonancia a su valor

máximo, de nuevo se obtiene el resultado deseado, obteniendo un pico de 12 db a la frecuencia de corte.

Fig. 4.16. – Señal de ruido blanco con filtrado paso alto y aumento del factor Q

Como último paso para comprobar el correcto funcionamiento del filtro se ha hecho una medición colocando el selector de tipo de filtro en su estado de filtro paso banda, para ello se han establecido valores para frecuencia de corte en la frecuencia de 1000 Hz que se esta utilizando como referencia y de 1 en el potenciómetro de resonancia. Tras esta medida se ha obtenido el resultado graficado en la siguiente imagen 4.17..

Fig. 4.17. – Señal de ruido blanco con filtrado paso banda

En ella se aprecia que el resultado del procesado es correcto, realizando un filtrado de las bandas inferior y superior respecto a la frecuencia de corte.

También se ha realizado una prueba aumentando el valor de resonancia a su valor máximo al utilizar el filtro paso banda, obteniendo el pico de 12 db en la frecuencia de corte que aparecía en las mediciones anteriores.

Fig. 4.18. – Señal de ruido blanco con filtrado paso banda y aumento del factor Q

4.5.2. Retardo en la señal y retroalimentación

El siguiente proceso de medición se ha realizado para comprobar el correcto funcionamiento del plugin dentro de la dimensión temporal. Para ello se ha utilizado el plugin gratuito de la compañía SmartElectronix llamado 'Exoscope', este software realiza la función de un osciloscopio, permitiendo analizar la señal en el eje temporal.

Para realizar estas medidas se ha utilizado una señal mucho mas impulsiva, correspondiente a un sonido percusivo de corta duración ya que permitirá ver el procesado temporal que sufre la señal de una forma mas correcta.

En la primera medida realizada se ha analizado al igual que en el caso del filtrado que el control de 'Dry/Wet', que permite realizar un bypass de la señal, funciona correctamente en el ámbito temporal. Para ello se ha situándolo en el valor 0 % donde solo deja pasar la señal de entrada y no añade procesado, este valor se ha comparado con una medición de la señal original sin pasar por el plugin desarrollado, obteniendo los siguientes resultados.

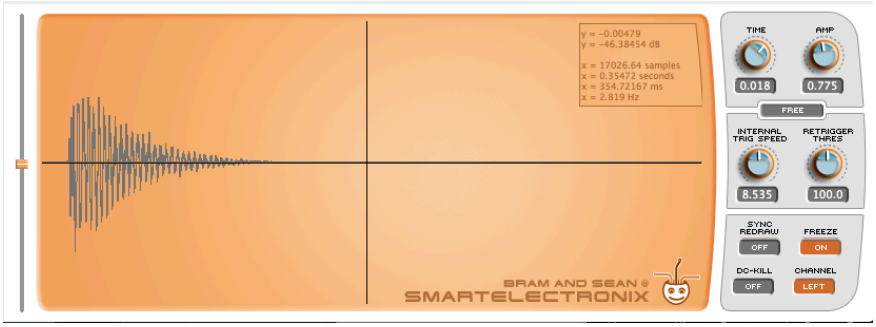

Fig. 4.19. – Señal impulsiva original

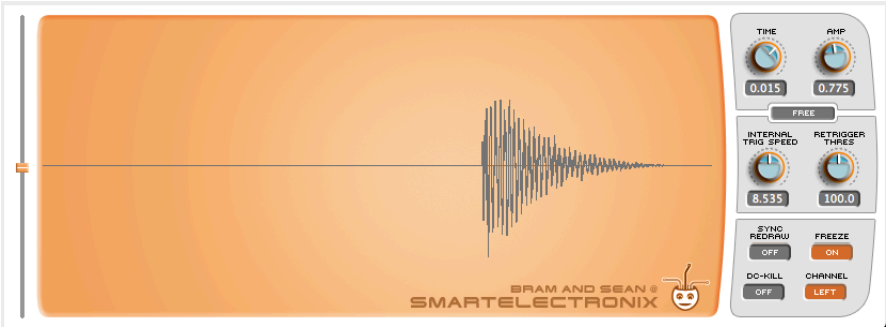

Fig. 4.20. – Señal pasada por el plugin sin procesado

Como se aprecia la señal mantiene la forma original intacta, por lo que en el ámbito temporal el control también funciona correctamente.

Tras esto para el resto de mediciones se ha situado el valor de 'Dry Wet' al 50 % para obtener en la misma proporción la señal de entrada y la señal procesada.

La siguiente medición se ha realizado utilizando el valor mínimo posible del potenciómetro 'Delay Time' que introduce un retardo temporal a la señal procesada de 50 ms. Para comprobar que su funcionamiento es correcto se ha tomado una muestra de la señal obtenida obteniendo los resultados que se muestran en la gráfica que aparece a continuación.

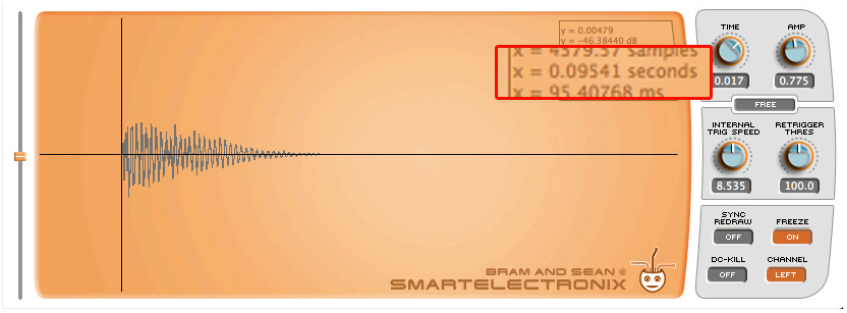

Fig. 4.21. – Instante temporal de inicio de la señal sin procesar

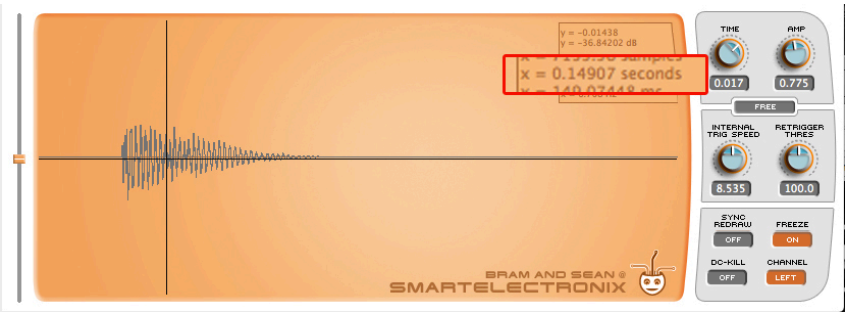

Fig. 4.22. – Instante temporal de inicio de la señal procesada

En las graficas se aprecia que el control funciona correctamente, ya que aparece una diferencia entre los picos iniciales de la señal y de su repetición de 50 ms, asumiendo que el

pequeño error que aparece es debido a que no se ha podido lograr mayor precisión en la toma de la medida.

Como siguiente medida en la dimensión temporal se ha tomado una muestra aumentando el valor de 'Delay Time' a su valor máximo de 500 ms, para comprobar que este control funciona correctamente en todo su rango de actuación. Tras esto se han obtenido las gráficas 4.23. y 4.24. que aparecen a continuación.

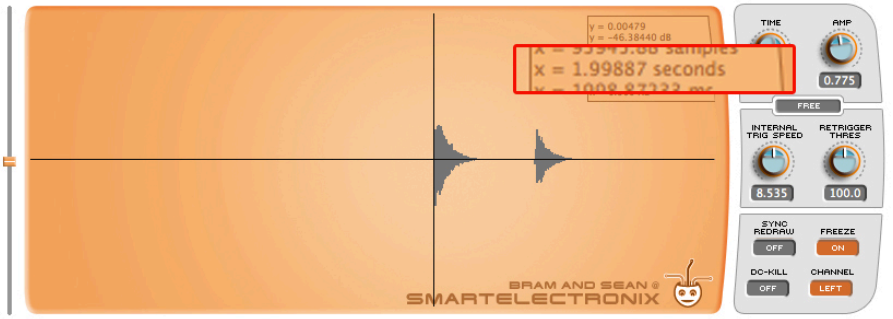

Fig. 4.23. – Instante temporal de inicio de la señal sin procesar

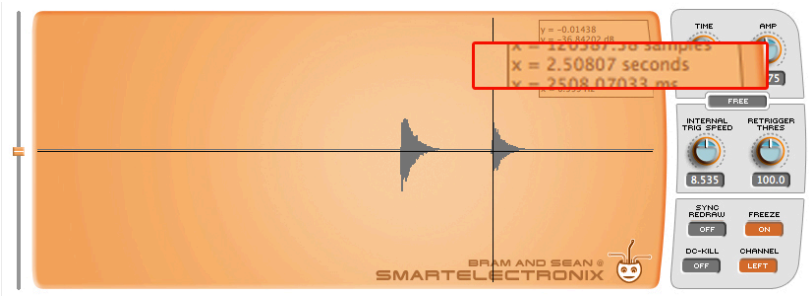

Fig. 4.24. – Instante temporal de inicio de la señal procesada

Vemos que aparecen marcados en rojo los instantes temporales donde aparece el pico de la señal original y su repetición y entre ellos hay una diferencia de 500 ms por lo que el funcionamiento de este parámetro es el deseado.

Como última prueba en el dominio temporal, se ha comprobado el correcto funcionamiento del parámetro de 'Feedback', para ello se ha mantenido el control de 'Delay Time' en su posición de 500 ms y se han tomado dos medidas. En la primera de ellas se ha mantenido 'Feedback' en su valor mínimo, y se ha obtenido el resultado que aparece en la siguiente figura 4.25..

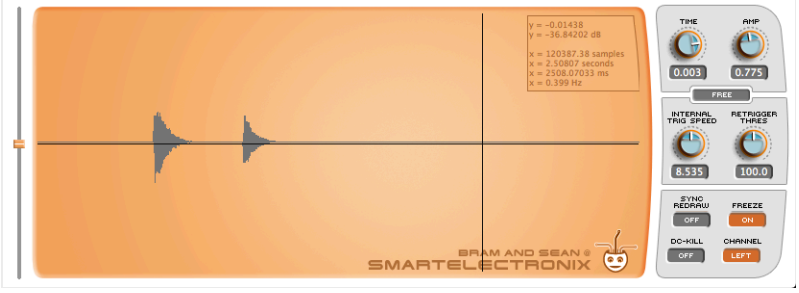

Fig. 4.25. – Representación temporal con feedback mínimo

En este caso la realimentación es nula, ya que el valor del potenciómetro es 0 % y no aparecen replicas de la señal procesada, por lo que su funcionamiento es correcto.

Para la última medida en el dominio temporal se ha posicionado el valor del potenciómetro del parámetro de 'Feedback' a su valor máximo, 95 %, donde la señal procesada se retroalimentará multiplicándose por un factor de 0,95 en cada nueva réplica, con lo que las replicas irán decayendo en amplitud de forma muy lenta. El resultado se puede ver en la siguiente figura.

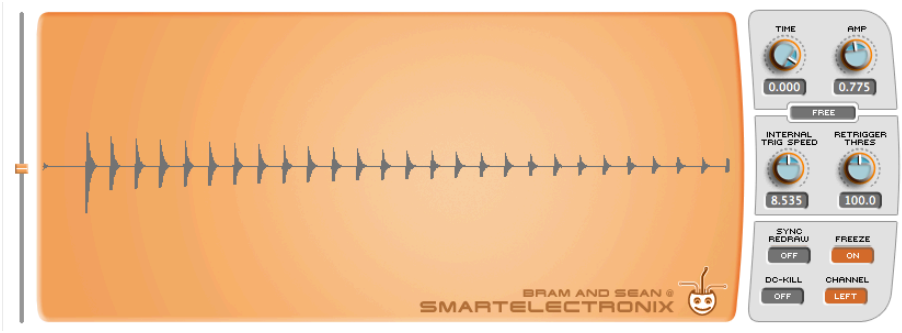

Fig. 4.26. – Representación temporal con feedback máximo

En la imagen 4.26. se aprecia que su funcionamiento es correcto, apareciendo un gran numero de réplicas de la señal procesada que van decayendo en el tiempo.

4.5.3. Posicionamiento espacial de la señal envolvente

Para comprobar el correcto funcionamiento de las rotaciones que sufre la señal procesada en el dominio espacial, en torno al eje z se han realizado mediciones con el plugin 'EnergyVisualizer' de IEM, que nos permite conocer la posición donde se encuentra la fuente de audio dentro del entorno espacial delimitado por una esfera.

Se han tomado varias muestras direccionales de la señal procesada por el plugin, para ello se ha establecido el parámetro de 'Angle' con un valor de 90º para tener una mayor precisión al comprobar que la rotación se realiza de la forma adecuada.

La primera muestra corresponde a la señal original, de la que se ha dejado pasar parte gracias al control 'Dry/Wet' del plugin. En la imagen siguiente 4.27. se puede apreciar su posición en el espacio de la esfera, en este caso la señal esta posicionada con unos valores de azimut de 0º y elevación 0º.

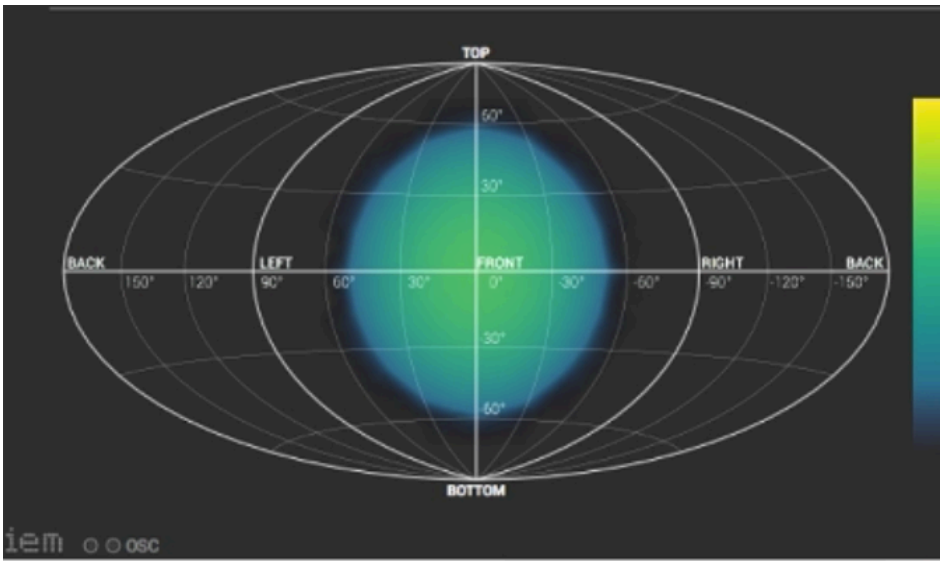

Fig. 4.27. – Posición de la señal original

La segunda muestra corresponde a la primera replica de la señal procesada, que ha sufrido el correspondiente filtrado, retardo respecto a la señal de entrada y rotación sobre el eje vertical *z*. En la siguiente imagen 4.28. se puede apreciar que la posición de esta segunda muestra es correcta, ya que el procesado ha introducido una rotación en azimut de 90º, situándola a la derecha del punto de escucha. De esta forma la señal comienza a rotar alrededor del oyente.

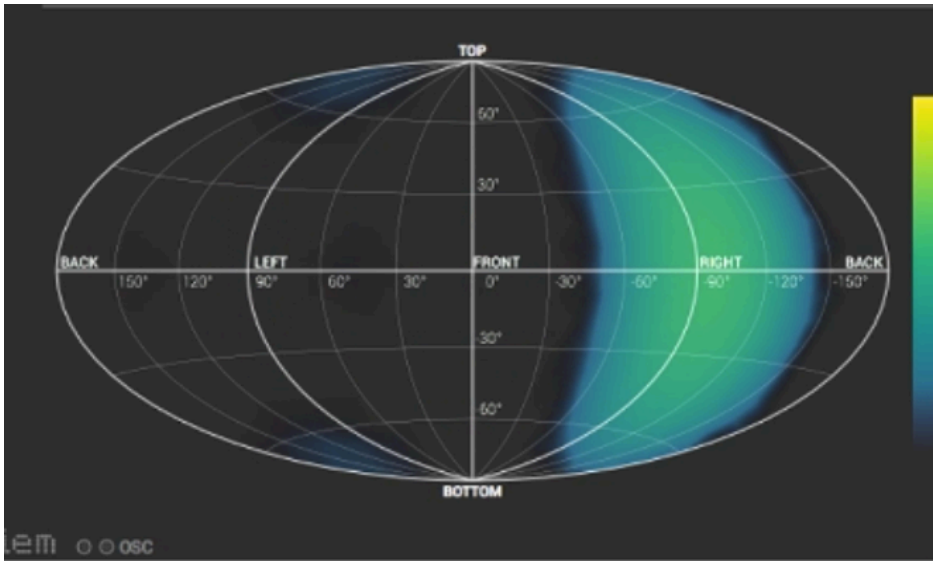

Fig. 4.28. – Posición de la señal de la primera replica

Por ultimo se a tomado una muestra de la primera réplica que sufre esta señal procesada gracias al control de 'Feedback'. En la siguiente imagen 4.29., se aprecia que con el procesado la señal continua rotando, añadiendo 90º mas a la posición de la muestra correspondiente a la primera aparición de la señal procesada, situándose en éste caso la réplica en la parte trasera con respecto a la posición de escucha.

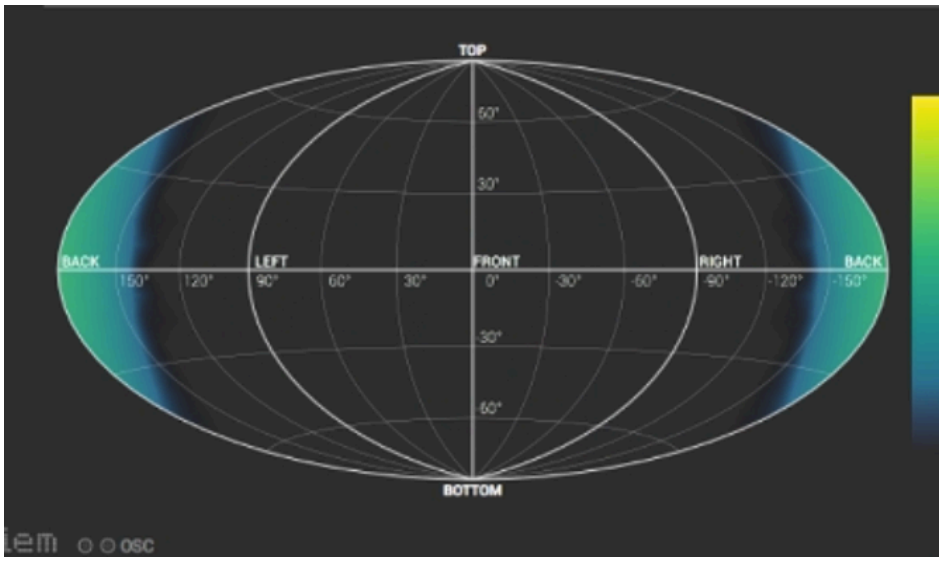

Fig. 4.29. – Posición de la señal de la segunda replica

Para el resto de réplicas el procesado continua, rotando de la misma manera, sumando 90º de azimut a la posición de cada nueva réplica aparecida. Con esta visualización gráfica, complementada con la apreciación auditiva se puede comprobar que el procesado que sufre la señal es el deseado.

4.5.4. Modificaciones de amplitud en la señal procesada

En este caso, se ha buscado comprobar que las variaciones de amplitud que sufren las replicas al aplicar mayor cantidad de retroalimentación con el control de 'Feedback' son correctas. Para ello se ha vuelto a utilizar el plugin 'Exoscope' que actúa como osciloscopio, donde se pueden apreciar con precisión las variaciones correspondientes a la dimensión de amplitud de la señal.

Para realizar esta comprobación se han graficado, la señal procesada retardada y su primera replica, colocando el valor de retroalimentación al 50 %, con esto el valor de cada replica se debería reducir multiplicándose por un factor de 0,5. El resultado se puede analizar en las siguientes imágenes 4.30. y 4.31..

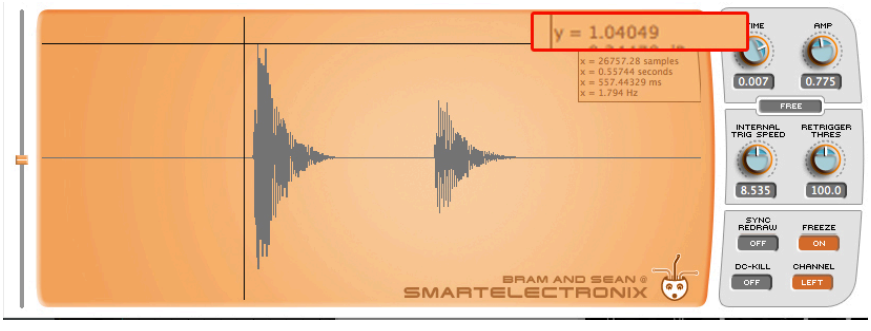

Fig. 4.30. – Valor de amplitud de la primera replica de la señal

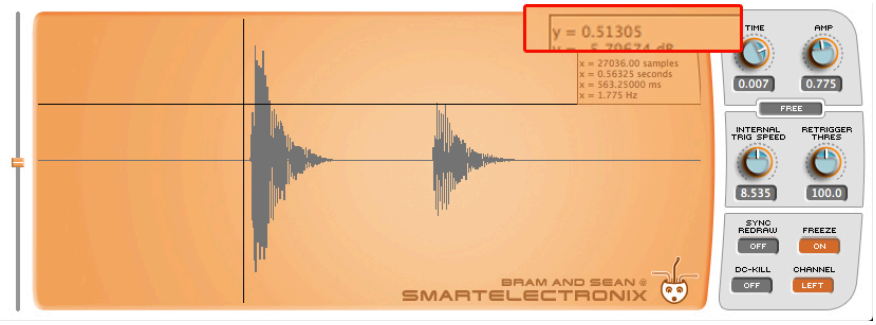

Fig. 4.31. – Valor de amplitud de la segunda replica de la señal

En estas representaciones gráficas se observa en el valor enmarcado en rojo que efectivamente la amplitud entre replicas se reduce en un 50 %, asumiendo que el pequeño error que aparece corresponde a las limitaciones de precisión en la toma de medidas.

Las ultimas medidas realizadas se corresponden a la comprobación del correcto funcionamiento del parámetro 'Output' que permite aumentar o reducir la ganancia de la señal de salida que proporciona el plugin tras el procesado.

En la primera gráfica 4.32. se puede observar enmarcado en rojo el valor de amplitud de la señal de entrada, que se tomara como referencia para ser comparada con las medidas posteriores.

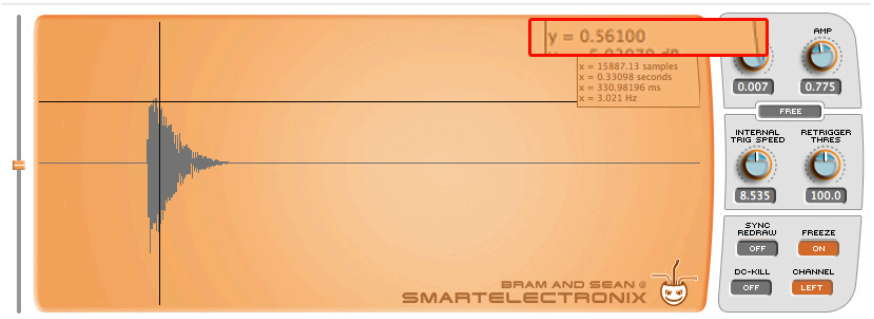

Fig. 4.32. – Amplitud de la señal sin pasar por el plugin

La siguiente 4.33. gráfica corresponde a la señal pasada por el plugin sin realizar ningún tipo de procesado. Esto permite ver que los valores de amplitud no cambian si se utiliza un bypass realizado gracias al parámetro 'Dry/Wet'. Para ello el control de 'Output' esta establecido con un valor de 1.0 por lo que la ganancia que se le aplica a la señal es nula.

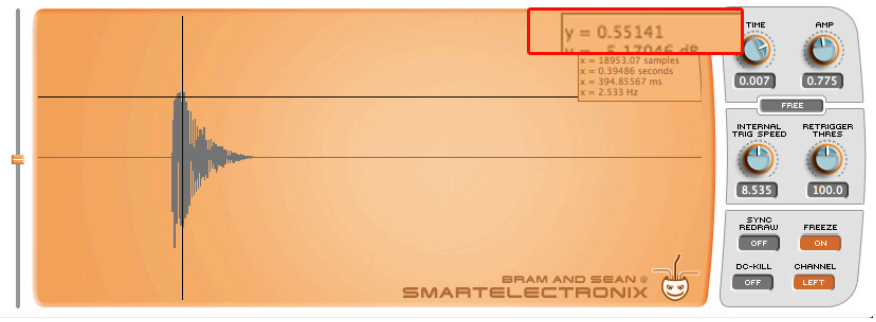

Fig. 4.33. – Amplitud de la señal pasada por el plugin sin procesado

Por ultimo se cuenta con una ultima medida, donde se esta aplicando una ganancia dada por el parámetro 'Output', establecido con un valor de 1,2. Esto aplica una ganancia tomando este valor como factor, por lo que el valor de amplitud aumenta en un 20 %. En la imagen siguiente 4.34. se puede ver que este ultimo procesado es correcto.

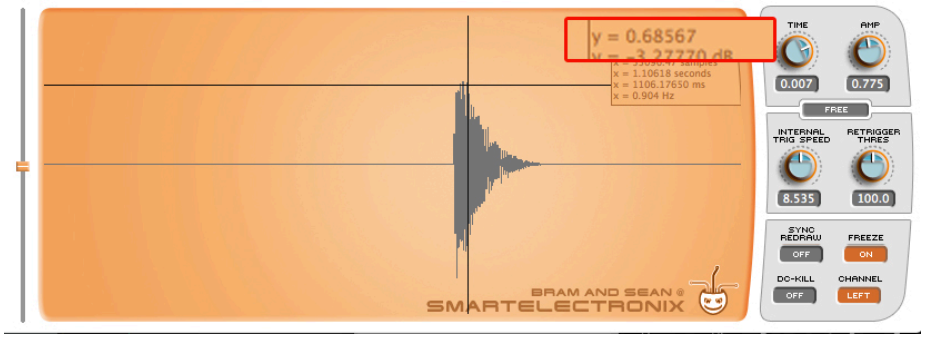

Fig. 4.34. – Amplitud de la señal con aumento de ganancia

5. CONCLUSIONES

Se ha conseguido realizar un trabajo funcional y aplicable dentro del entorno del proyecto Sonorizarte, obteniendo una aplicación en forma de plugin de audio que podrá ser utilizada por los productores, artistas y diseñadores sonoros que participarán en los trabajos de producción de audio destinados a ser reproducidos en la configuración de altavoces que se implementará.

Por otro lado se trata de una aplicación perfectamente funcional dentro de cualquier producción de audio Ambisonic fuera del proyecto sonorizarte, por lo que se trata de un software con aplicación en el mundo real y que podrá ser utilizado por quien lo desee.

A través de la realización de este proyecto, también se ha conseguido obtener un conocimiento en profundidad de las señales Ambisonic, consiguiendo conocer así los métodos de codificación y decodificación, el flujo de señal para la creación de una producción de audio envolvente y la generación de efectos para transformaciones de la escena sonora tridimensional.

También se ha realizado un análisis, para su conocimiento, de las técnicas de producción de audio actuales, del software que es utilizado habitualmente, centrando este análisis en Reaper, ya que se trata de la estación de trabajo de audio mas adaptada a las necesidades del trabajo con señales Ambisonic. Además se ha profundizado en el uso de plugins para el procesado de audio y conocer los tipos que existen.

De la misma forma se ha profundizado en el entorno de programación orientado al desarrollo de plugins de audio Juce, basado en C++, consiguiendo un gran conocimiento y fluidez de trabajo dentro de este framework y con las técnicas de programación orientadas al procesado de audio.

Además se ha investigado sobre el estado actual comercial del campo de los plugins de audio aplicables en el mundo de Ambisonics al realizar un estudio del estado del arte. Esto ha permitido realizar un análisis de estos plugins y aprender a utilizarlos dentro de un entorno de producción de audio.

Aparte de conseguir cumplir con todos estos objetivos, se ha realizado un proceso de análisis de los resultados obtenidos a raíz de este proyecto en forma de mediciones de las señales resultantes al procesar audio con el software desarrollado, concluyendo y asegurando así que los resultados son los deseados y que el plugin cumple con los objetivos establecidos.

En lo personal, este proyecto me ha servido para introducirme en el campo de las señales Ambisonics, ya que es una tecnología en la que me parece interesante profundizar porque se encuentra en pleno desarrollo gracias a los sistemas de audio inmersivo y las aplicaciones que están desarrollando diferentes compañías punteras como podrían ser Facebook, Google o Youtube y la alta demanda que están teniendo sistemas de este tipo.

Por otro lado este trabajo me ha permitido introducirme en el desarrollo y programación de plugins de audio. Este es un campo en el que tengo especial interés porque he estado varios años trabajando y estudiando temas de producción de audio en los que se trabaja habitualmente procesando las señales a través de plugins.

Por ultimo, recalcar que este proyecto me ha permitido realizar una aplicación directa de la metodología proyectual que he obtenido durante el estudio del grado, además de aplicar técnicas proyectuales de otros estudios que he realizado, como podría ser el campo del diseño gráfico. Además, dentro del trabajo, ha habido una aplicación de conocimientos, obtenidos durante estos cuatro años del grado, muy amplia, aplicando en él numerosos conceptos correspondientes a los campos principales del procesado de señal y la programación, además de los específicos de la rama en la que me especializado orientada al procesado de audio e imagen.

6. BIBLIOGRAFÍA

A continuación se incluye la lista de referencias principales que han sido utilizadas para la realización de este proyecto.

Stroustrup, Bjarne (1997). *The C++ Programming Language 3rd Ed.* AT&T.

Kronlachner, Matthias (2014). *Spatial Transformations for the Alterarion of Ambisonic Recordings.* Austria: Institute of Electronic Music and Acoustics University of Music and Performing Arts.

Robinson, Martin (2013). *Getting Started with Juce.* Packt Publishing.

Zotter, Franz y Frank, Matthias (2019). *Ambisonics: A Practical 3D Audio Theory for Recording, Studio Production, Sound Reinforcement, and Virtual Reality.* Austria: Institute of Electronic Music and Acoustics University of Music and Performing Arts.

Zotter, Franz y Frank, Matthias (2019). *Spatial transformations for the enhancement of Ambisonic recordings.* Austria: Institute of Electronic Music and Acoustics University of Music and Performing Arts.